

Extending Dreamweaver

Trademarks

Afterburner, AppletAce, Attain, Attain Enterprise Learning System, Attain Essentials, Attain Objects for Dreamweaver, Authorware, Authorware Attain, Authorware Interactive Studio, Authorware Star, Authorware Synergy, Backstage, Backstage Designer, Backstage Desktop Studio, Backstage Enterprise Studio, Backstage Internet Studio, Design in Motion, Director, Director Multimedia Studio, Doc Around the Clock, Dreamweaver, Dreamweaver Attain, Drumbeat, Drumbeat 2000, Extreme 3D, Fireworks, Flash, Fontographer, FreeHand, FreeHand Graphics Studio, Generator, Generator Developer's Studio, Generator Dynamic Graphics Server, Knowledge Objects, Knowledge Stream, Knowledge Track, Lingo, Live Effects, Macromedia, Macromedia M Logo & Design, Macromedia Flash, Macromedia Xres, Macromind, Macromind Action, MAGIC, Mediamaker, Object Authoring, Power Applets, Priority Access, Roundtrip HTML, Scriptlets, SoundEdit, ShockRave, Shockmachine, Shockwave, Shockwave Remote, Shockwave Internet Studio, Showcase, Tools to Power Your Ideas, Universal Media, Virtuoso, Web Design 101, Whirlwind and Xtra are trademarks of Macromedia, Inc. and may be registered in the United States or in other jurisdictions including internationally. Other product names, logos, designs, titles, words or phrases mentioned within this publication may be trademarks, servicemarks, or tradenames of Macromedia, Inc. or other entities and may be registered in certain jurisdictions including internationally.

This guide contains links to third-party Web sites that are not under the control of Macromedia, and Macromedia is not responsible for the content on any linked site. If you access a third-party Web site mentioned in this guide, then you do so at your own risk. Macromedia provides these links only as a convenience, and the inclusion of the link does not imply that Macromedia endorses or accepts any responsibility for the content on those third-party sites.

Apple Disclaimer

APPLE COMPUTER, INC. MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, REGARDING THE ENCLOSED COMPUTER SOFTWARE PACKAGE, ITS MERCHANTABILITY OR ITS FITNESS FOR ANY PARTICULAR PURPOSE. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY PROVIDES YOU WITH SPECIFIC LEGAL RIGHTS. THERE MAY BE OTHER RIGHTS THAT YOU MAY HAVE WHICH VARY FROM STATE TO STATE.

Copyright © 2000 Macromedia, Inc. All rights reserved. This manual may not be copied, photocopied, reproduced, translated, or converted to any electronic or machine-readable form in whole or in part without prior written approval of Macromedia, Inc.

Acknowledgments

Project Management: Sheila McGinn

Writing: JuLee Burdekin, David Jacowitz, and Denise Lee

Editing: Susan Moxley

Multimedia Design and Production: Aaron Begley and Noah Zilberberg

Print Production: Chris Basmajian, Paul Benkman, Caroline Branch, and Rebecca Godbois

Web Editing: Jane Flint DeKoven and Jeff Harmon

Special thanks to Heidi Bauer, Winsha Chen, Chris Denend, Margaret Dumas, Peter Fenczik, Dave George, Valerie Green, Joel Huff, Lori Hylan, Narciso (nj) Jaramillo, Sho Kuwamoto, Jay London, Joe Marini, Charles McBrian, Jeff Schang, Ken Sundermeyer, and the Dreamweaver and Dreamweaver UltraDev engineering and QA teams.

First Edition: November 2000

Macromedia, Inc.
600 Townsend St.
San Francisco, CA 94103

CONTENTS

CHAPTER 1

Extending Dreamweaver Overview	7
Customizing or extending?	7
Reading this book	8
The extension architecture	9
Installing an extension	11
Errata	12
Conventions used in this guide	12

CHAPTER 2

The Document Object Model and JavaScript	13
The document object model in Dreamweaver	14
How JavaScript works in extensions	23
Custom JavaScript controls	24

CHAPTER 3

Objects	31
How object files work	32
The object API	33
Adding objects to the Objects panel	36
Adding objects to the Insert menu	36

CHAPTER 4

Commands	37
How commands work	38
The command API	39
A simple command example	42
Adding commands to the Commands menu	44

CHAPTER 5	
Menu Commands	45
How menu commands work	46
The menu command API	47
A simple menu command	51
A simple dynamic menu	53
CHAPTER 6	
Reports	55
How reports work	56
The report API	57
CHAPTER 7	
JavaScript Debugger Modules	61
CHAPTER 8	
Property Inspectors	69
How Property Inspector files work	71
The Property inspector API	72
A simple Property inspector example	74
CHAPTER 9	
Floating Panels	77
How Floating Panel files work	78
The floating panel API	79
About performance	83
A simple floating panel example	85
CHAPTER 10	
Behaviors	87
How behaviors work	88
The behavior API	89
A simple behavior example	98
CHAPTER 11	
The Fireworks Integration API	101
The Fireworks integration API	101
A simple Fireworks integration example	106

CHAPTER 12	
The Flash Objects API	109
CHAPTER 13	
The Design Notes API	115
How Design Notes work	116
The Design Notes JavaScript API	116
The Design Notes C API	121
CHAPTER 14	
The File I/O API	129
Verifying that DWfile is installed	129
The file I/O API	130
CHAPTER 15	
The HTTP API	137
CHAPTER 16	
The Database API	145
CHAPTER 17	
The JavaBean API	161
CHAPTER 18	
The Source Control Integration API	165
The source control integration API required functions	166
The source control integration API optional functions	173
Enablers	184
CHAPTER 19	
C-Level Extensibility	191
CHAPTER 20	
The Dreamweaver JavaScript API	205
Getting document data through the DOM	207
Assets panel functions	208
Behavior functions	216
Clipboard functions	227
Command functions	233
Conversion functions	234

CSS style functions	236
External application functions.	243
File manipulation functions	250
Find/replace functions.	260
Frame and frameset functions	266
General editing functions	268
Global application functions	287
Global document functions.	289
History functions	291
HTML style functions	300
JavaScript Debugger functions	304
Keyboard functions.	308
Layer and image map functions.	316
Library and template functions	320
Menu functions.	328
Path functions.	330
Quick Tag Editor functions.	333
Report functions	336
Results window functions	338
Selection functions	341
Site functions	348
Source View functions.	372
String manipulation functions.	388
Table editing functions	393
Timeline functions	402
Toggle functions	409
Translation functions.	431
Layout environment functions	434
Layout view functions.	441
Window functions	447
Deprecated functions	456
Enablers	465

INDEX	497
------------------------	------------

CHAPTER 1

Extending Dreamweaver Overview

This book covers the advanced capabilities you get when you extend Dreamweaver. Extensions are objects, commands, menu commands, panels, data translators, Property inspectors, reports, and behaviors that you create using the Dreamweaver application programming interface (API). This book helps you write your own extensions by providing information about how to program each type of extension and by explaining the Dreamweaver API. Read this chapter to find out the requirements involved in writing extensions, as well as for an overview of extensions.

Customizing or extending?

Before you begin reading this book, take a look at “Customizing Dreamweaver,” in the Dreamweaver 4 user guide. This chapter provides procedures for changing Dreamweaver panels, menus, dialog boxes, and HTML formats. It also explains some of the basics involved in extending Dreamweaver functionality, including how to edit Dreamweaver commands and how to add third-party tags.

When you’re ready to extend Dreamweaver—to create your own objects, commands, menu commands, panels, Property inspectors, reports, data translators, and behaviors—read this book.

Note: Before you begin writing extensions, refer to “Using Dreamweaver” for more information on how to install Dreamweaver.

Reading this book

This book describes the supported Dreamweaver API. Dreamweaver extensions are written in JavaScript. The scripts can perform edits on the document using a Document Object Model (DOM), and they can call C code that you have packaged in a DLL. You should have some experience in creating Web pages with Dreamweaver. Also, you should be familiar with the languages you use to extend Dreamweaver, either JavaScript or C.

The following sections describe the chapters in the order they appear in this book.

Looking at the big picture

- This chapter outlines the Dreamweaver extension architecture and how to begin creating extensions.
- “The Document Object Model and JavaScript in Extensions” explains the Dreamweaver Document Object Model (DOM) and how to access and display data from the user’s document through JavaScript calls to the document’s DOM.

Developing extensions

- “Objects” describes object extensions, details the object API, explains how the object file works, and provides examples.
- “Commands” explains how commands work and how to add your command to a menu. This chapter also details the commands API and provides examples.
- “Menu Commands” explains how menu commands work and how to add your command to a menu. This chapter also details the menu commands API and provides examples.
- “Reports” explains how to extend the set of prewritten reports shipped with Dreamweaver.
- “Debugger” explains how to extend the JavaScript debugger, create a debug version of the user’s document, step through the JavaScript code, and return errors.
- “Property Inspectors” describes how to create custom inspectors, details the Property inspector API, explains how the inspectors work, and provides examples.
- “Floating Panels” describes how floating panel extensions work, details the floating panel API, and provides examples.
- “Behaviors” explains how to write a behavior action, details the behavior API, explains how behaviors work, and provides examples.

Working with other products and utilities

- “Fireworks Integration API” explains how to launch and execute commands in Fireworks from Dreamweaver.
- “Flash Objects” explains how to create and modify Flash Object files.
- “The Design Notes API” describes the JavaScript and C functions that add file information, such as document history and association, to Design Notes.
- “The File I/O API” explains common file manipulation methods available through the `DWfile` object.
- “The HTTP API” explains how to get and post files to an HTTP server using the `DWHttpRequest` object.
- “The Source Control API” explains how to integrate source control application with Dreamweaver.
- “Data Translators” describes how translators work, details the data translator API, and how to create a translator, lock tags, and attributes.
- “C-Level Extensibility” explains how to create an interface between your C libraries and the Dreamweaver JavaScript API.
- “The Dreamweaver JavaScript API” is your reference guide to the main Dreamweaver API. It covers the JavaScript API according to functionality.

The extension architecture

The open nature of the Dreamweaver architecture enables you to modify virtually every aspect of Dreamweaver. How you interface with the product depends on the type of extension you write. This section explains the types of extension you can create. For additional information regarding the Dreamweaver architecture, read “The Document Object Model and JavaScript” on page 13, which explains the Dreamweaver DOM and how JavaScript works with extensions.

Extension types

A simple way to extend Dreamweaver is to add a button on the object panel. Then, each time the user drags that object onto the Design view of the Document window, Dreamweaver inserts the code associated with the object into the HTML. You create this kind of select-to-insert functionality by writing an object.

The following list represents the different ways you can extend Dreamweaver. You can extend Dreamweaver in one or all of the following ways: add objects to the Objects panel, add commands that insert or rearrange HTML tags and attributes, create your own floating panel, add client-side JavaScript behaviors, or write translators that convert specialized code into standard HTML.

The different extension types can be summarized as follows:

- 1 **Object**—Also known as an “insert-only” extension. You write this type of extension to add a new object to the Objects panel. You create this type of extension by writing an HTML file that contains the code to be inserted into the document. It can also contain a form that gathers input from the user and JavaScript that processes the input. You place the HTML file in the Configuration/Objects/*folder (where * is a wildcard representing any subfolder) along with an icon in GIF format, and the extension appears on the Objects panel.
- 2 **Command**—A command is a menu item that invokes a script. When you create this type of extension, you add a new menu item to Dreamweaver. To create a command, you create a file that implements the functions in the commands API (see “Commands” on page 37). If you want the command to appear in the “Commands” menu, just install the file in the Configuration/Commands folder. If you want the command to appear elsewhere in the menu system, you can customize the menu.xml file. When the user pops up a menu, the command can specify whether its menu item is grayed out or not.
- 3 **Panel**—Dreamweaver has panels that provide information about the current document and the current selection. You can add your own floating panel that interacts with the selection, the document, or the task, or that simply displays useful information. Floating panel files are files that contain HTML and JavaScript, and that live in the Configuration/Floaters folder.
- 4 **Inspector**—Inspectors are indispensable for defining, reviewing, and changing the name, size, appearance, and other attributes of the selection, as well as for launching internal and external editors for the selected element. Dreamweaver has several built-in interfaces for the Property inspector that let you set properties for many standard HTML tags. With custom property inspector files, you can override these built-in interfaces or create new ones to inspect custom tags.
- 5 **Behavior**—You can add new behaviors to the Plus (+) menu in the Behaviors panel. A behavior is a user event plus an action. You write the action. To create a behavior, you write behavior code, create a user interface in HTML to get input, create a handler to the event to which you want your behavior associated, determine if your behavior applies to a given document, and apply the behavior in the appropriate location. The Configuration/Behaviors/Actions folder contains many examples of behaviors.
- 6 **Data Translator**—A translator provides a visual representation of non-HTML code in the Design view of the Document window. It converts non-HTML code into HTML and locks the non-HTML code to prevent it from being parsed by Dreamweaver. You create translator extensions to represent non-HTML code in the Design view of the Document window.

Extension folders

By familiarizing yourself with the folders under the Configuration folder, you can discover the API for the areas you'll be extending and find working examples of each extension type.

Tip: The Configuration_ReadMe.htm file in the Configuration folder provides details on the contents of each subfolder.

One folder that does not correspond to a particular extension type is the Shared folder. The Shared folder is the central repository for utility functions, classes, and images that are commonly used by all extensions. For example, files in the Shared/MM/scripts/CMN folder contain functions that search all children of a node for a specific tag, strip spaces from a string, and generate the correct JavaScript reference to an object given its name, among other useful tasks.

Installing an extension

Before you program your own extension, download and install a pre-existing extension from the Macromedia Exchange Web site (<http://macromedia.com/exchange/>). Taking time to do so helps in a couple of ways. You can get a good idea of how many extensions programmers there are in the community. You may have need for a particular extension that has been created already, or that is close enough to review and revise for your own purposes.

Once you find an extension you are interested in on the Dreamweaver Exchange site, installing an extension is easy.

To install an extension, take the following steps:

- 1 If you haven't already done so, install the Package Manager. You can download it from the Macromedia Exchange Web site (<http://www.macromedia.com/software/downloads/>).
- 2 Log on to the Macromedia Exchange Web site (<http://www.macromedia.com>).
- 3 From the Macromedia Exchange Web site, select an extension you'd like to add. Pick a simple extension, one that will not drastically alter your Dreamweaver development environment. Click the download link on the extension detail page to download the extension package.
- 4 Save the extension package in the Downloaded Extensions folder of your installed Dreamweaver folder.
- 5 From the Package Manager File menu, select "Install Extension" (Commands > Manage Extensions).

The extension is automatically installed in Dreamweaver.

Some extensions need Dreamweaver to restart before you can access them. If you are running Dreamweaver when you install the extension, you may be prompted to quit and restart the application.

Go to the Package Manager (Commands > Manage Extensions) in Dreamweaver to view basic information on the extension (for example, where to access it within Dreamweaver).

Errata

This documentation was written before the code in Dreamweaver was complete. Thus, there may be some discrepancies between the final implementation of the JavaScript API in Dreamweaver and how it is documented in this book. A list of known issues can be found in the Extensibility section of the Dreamweaver Support Center at <http://www.macromedia.com/support/dreamweaver/extend.html>.

Conventions used in this guide

The following typographical conventions are used in this guide:

- Code font indicates code fragments and API literals, including class names, method names, function names, type names, scripts, SQL statements, and HTML tag and attribute names.
- *Italic code* font indicates replaceable items in code.
- The continuation symbol (–) indicates that a long line of code has been broken across two or more lines. Due to margin limits in this book's format, what is otherwise a continuous line of code must be split. When you see this continuation symbol in this book, consider the subsequent line to be a part of the first. When copying the lines of code, type the lines as one line.

The following naming conventions are used in this guide:

- You—the developer responsible for writing extensions.
- The user—the person using Dreamweaver.
- The visitor—the person who views the Web page as created by the user.

CHAPTER 2

The Document Object Model and JavaScript

.....

HTML documents consist of a tree of tags that reveals the document's structure. The root of the tree is the HTML tag. The two largest branches of the tree are HEAD and BODY. Offshoots of HEAD include TITLE, STYLE, SCRIPT, ISINDEX, BASE, META, and LINK. Offshoots of BODY include headings (H1, H2, and so on), block-level elements (P, DIV, FORM, and so on), text-level elements (FONT, BR, IMG, and so on), and the ADDRESS element. Leaves on the offshoots include attributes such as WIDTH, HEIGHT, ALT, and HREF.

A document object model, or DOM, is also a tree that discloses the document's structure. The DOM, however, reports this structure in terms of objects and properties, rather than in terms of tags and attributes.

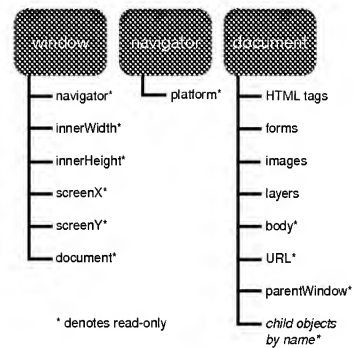
The root of the DOM tree is the document itself, the HTML object is the trunk, and the rest of the objects in the document branch from the HTML object as the HTML tags and attributes do.

The document object model in Dreamweaver

A browser's DOM determines how the JavaScript in an HTML document works in that browser. Similarly, Dreamweaver DOM determines how the JavaScript in extensions works in Dreamweaver.

Dreamweaver DOM combines a subset of the Netscape Navigator 4.0 DOM with a subset of the World Wide Web Consortium (W3C) DOM Level 1. With the incorporation of DOM Level 1, every part of an HTML page has become an object—including tags (which the W3C calls elements), comments, and text.

The basic breakdown of the DOM hierarchy is as follows:



Objects can be referred to by index (`document.forms[3].elements[1]`) or by name (`document.myForm.myButton`). Objects with the same name are collapsed into an array. You can access a particular object in the group by index (for example, the first radio button with the name `myRadioGroup` in `myForm` would be referenced as `document.myForm.myRadioGroup[0]`).

The following table gives an overview of the properties, methods, and events supported by each object; these are described in more detail in books such as *JavaScript: The Definitive Guide* (O'Reilly). Additional details about the W3C properties and methods, which are less thoroughly documented by third parties, follow the table. A bullet (•) marks read-only properties.

Object	Properties	Methods	Events
window	document • navigator • innerWidth • innerHeight • screenX • screenY •	alert() confirm() escape() unescape() close() setTimeout() clearTimeout() setInterval() clearInterval() resizeTo()	onResize
navigator	platform •	None	None
document	forms • (an array of form objects) images • (an array of image objects) layers • (an array of LAYER, ILAYER, and absolutely positioned DIV and SPAN objects) <i>child objects by name</i> • nodeType • parentNode • childNodes • documentElement • body • URL • parentWindow •	getElementsBy TagName() hasChildNodes()	onLoad
all tags/elements	nodeType • parentNode • childNodes • tagName • <i>attributes by name</i> innerHTML outerHTML	getAttribute() setAttribute() removeAttribute () getElementsByTa gName() hasChildNodes()	

Object	Properties	Methods	Events
form	In addition to the properties available for all tags: elements • (an array of button, checkbox, password, radio, reset, select, submit, text, file, hidden, image, and textarea objects) mmcolorbutton child objects by name •	Only those methods available to all tags.	None
layer	In addition to the properties available for all tags: visibility left top width height zIndex	Only those methods available to all tags.	None
image	In addition to the properties available for all tags: src	Only those methods available to all tags.	onMouseOver onMouseOut onMouseDown onMouseUp
button reset submit	In addition to the properties available for all tags: form •	In addition to the methods available for all tags: blur() focus()	onClick
checkbox radio	In addition to the properties available for all tags: checked form •	In addition to the methods available for all tags: blur() focus()	onClick
password text file hidden image (field) textarea	In addition to the properties available for all tags: form • value	In addition to the methods available for all tags: blur() focus() select()	onBlur onFocus

Object	Properties	Methods	Events
select	In addition to the properties available for all tags: form • options • (an array of option objects) selectedIndex	In addition to the methods available for all tags: blur() (Windows only) focus() (Windows only)	onBlur (Windows only) onChange onFocus (Windows only)
option	In addition to the properties available for all tags: text	Only those methods available to all tags.	None
mmcolorbutton	In addition to the properties available for all tags: name value	None	onChange
array	Matches Netscape 4	Matches Netscape 4	None
boolean			
date			
function			
math			
number			
object			
string			
regexp			
text	nodeType • parentNode • childNodes • data	hasChildNodes()	None
comment	nodeType • parentNode • childNodes • data	hasChildNodes()	None
NodeList	length •	item()	None
NamedNodeMap	length •	item()	None

The dreamweaver object and its properties

Dreamweaver implements the standard objects as defined by the browsers and the W3C, as well as two custom objects: `dreamweaver` and `site`. The `dreamweaver` object has two read-only properties associated with it: `appName` and `appVersion`.

`appName` has the value "Dreamweaver" or "Dreamweaver Ultradev" for each respective application. `appVersion` has a value of the form "*versionNumber* [*languageCode*] (*platform*)". For example, the value of the `appVersion` property for the Swedish Windows version of Dreamweaver 4 would be "4.0 [se] (Win32)"; the value for the English Macintosh version would be "4.0 [en] (MacPPC)".

The `appName` and `appVersion` properties were implemented in Dreamweaver 3 and are not available in earlier versions of Dreamweaver. To determine whether the version of Dreamweaver is 3 or later, you can simply check for the existence of the `appVersion` or `appName` property. To check for a specific version of Dreamweaver, check first for the existence of `appVersion` and then for the version number. For example:

```
if (dreamweaver.appVersion &&
    dreamweaver.appVersion.indexOf('3.01') != -1){
    // execute code
}
```

The `site` object has no properties. For more information about the methods of the `dreamweaver` and `site` objects, see "The Dreamweaver JavaScript API" on page 205.

DOM details

Unlike the Netscape DOM, DOM Level 1 has not been documented in hundreds of third-party books and Web sites. Thus, this document will describe the DOM Level 1 properties and methods, and the values they return, in some detail.

DOM Level 1 introduces four constants that describe the types of objects that make up the document tree (called *nodes*). These constants, which usually show up as return values for the `nodeType` property, are:

```
Node.DOCUMENT_NODE
Node.ELEMENT_NODE
Node.COMMENT_NODE
Node.TEXT_NODE
```

Properties and methods of the document object

The following table lists the new properties and methods of the document object in Dreamweaver, along with their return values (with explanations, where appropriate). A bullet (•) marks read-only properties.

Property or method	Return value and explanation
nodeType •	Node.DOCUMENT_NODE
parentNode •	null
parentWindow •	The JavaScript object corresponding to the document's parent window. (This property is not included in DOM Level 1; however, it is supported by Microsoft Internet Explorer (IE) 4.0.)
childNodes •	A NodeList containing all the immediate children of the document object. Typically the document will have a single child: the HTML object.
documentElement •	The JavaScript object corresponding to the HTML tag. This property is shorthand for getting the value of document.childNodes and extracting the HTML tag from the NodeList.
body •	The JavaScript object corresponding to the BODY tag. This property is shorthand for calling document.documentElement.childNodes and extracting the BODY tag from the NodeList. For frameset documents, this property returns the node for the outermost frameset instead.
URL •	The file:// URL for the document or, if the file has not been saved, an empty string.
getElementsByTagName(tagName)	<p>A NodeList that can be used to step through tags of type <i>tagName</i> (for example, IMG, DIV, and so on).</p> <p>If the <i>tag</i> argument is LAYER, the function returns all LAYER and I LAYER tags and all absolutely positioned DIV and SPAN tags.</p> <p>If the <i>tag</i> argument is INPUT, the function returns all form elements. (For this shortcut to work properly, all form field names must begin with a letter.)</p>
hasChildNodes()	true

Properties and methods of HTML tag objects

Every HTML tag is represented by a JavaScript object. Tags are organized in a tree hierarchy, where tag *x* is a parent of tag *y* if *y* falls completely within *x*'s opening and closing tags (*<x>x content surrounding <y>y content</y></x>*). The following table lists the properties and methods of tag objects in Dreamweaver, along with their return values (with explanations, where appropriate). A bullet (•) marks read-only properties.

Property or method	Return value and explanation
<code>nodeType</code> •	<code>Node.ELEMENT_NODE</code>
<code>parentNode</code> •	The parent tag. If this is the HTML tag, then the document object is returned instead.
<code>childNodes</code> •	A <code>NodeList</code> containing all the immediate children of the tag.
<code>tagName</code> •	The HTML name for the tag, such as <code>IMG</code> , <code>A</code> , or <code>BLINK</code> . This value is always returned in uppercase letters.
<code>attrName</code>	A string containing the value of the specified tag attribute. <code>tag.attrName</code> cannot be used if <code>attrName</code> is a reserved word in the JavaScript language (for example, <code>class</code>). In this case, use <code>getAttribute()</code> and <code>setAttribute()</code> instead.
<code>innerHTML</code>	The HTML source code contained between the begin tag and the end tag. For example, in the code <code><p>Hello, World!</p></code> , <code>p.innerHTML</code> would return <code>Hello, World!</code> . If you write to this property, the DOM tree is immediately updated to reflect the new structure of the document. (This property is not included in DOM Level 1; however, it is supported by IE 4.0.)
<code>outerHTML</code>	The HTML source code for this tag, including the tag itself. For the example code above, <code>p.outerHTML</code> would return <code><p>Hello, World!</p></code> . If you write to this property, the DOM tree is immediately updated to reflect the new structure of the document. (This property is not included in DOM Level 1; however, it is supported by IE 4.0.)
<code>getAttribute(attrName)</code>	The value of the specified attribute if it is explicitly specified; otherwise, <code>null</code> .

Property or method	Return value and explanation
<code>getTranslatedAttribute(<i>attrName</i>)</code>	The translated value of the specified attribute, or the same value that would be returned by <code>getAttribute()</code> if the attribute's value is not translated. (This property is not included in DOM Level 1; it was added to Dreamweaver 3 to support attribute translation.)
<code>setAttribute(<i>attrName</i>, <i>attrValue</i>)</code>	No return value. Sets the specified attribute to the specified value: for example, <code>img.setAttribute("src", "image/roses.gif")</code> .
<code>removeAttribute(<i>attrName</i>)</code>	No return value. Removes the specified attribute and its value from the HTML for this tag.
<code>getElementsByTagName(<i>tagName</i>)</code>	A <code>NodeList</code> that can be used to step through child tags of type <i>tagName</i> (for example, <code>IMG</code> , <code>DIV</code> , and so on). If the <i>tag</i> argument is <code>LAYER</code> , the function returns all <code>LAYER</code> and <code>ILAYER</code> tags and all absolutely positioned <code>DIV</code> and <code>SPAN</code> tags. If the <i>tag</i> argument is <code>INPUT</code> , the function returns all form elements. (For this shortcut to work properly, all form field names must begin with a letter.)
<code>hasChildNodes()</code>	A Boolean value indicating whether the tag has any children.
<code>hasTranslatedAttributes()</code>	A Boolean value indicating whether the tag has any translated attributes. (This property is not included in DOM Level 1; it was added to Dreamweaver 3 to support attribute translation.)

Properties and methods of text objects

Each contiguous block of text in an HTML document (for example, the text within a `P` tag) is represented by a JavaScript object. Text objects never have children. The following table lists the properties and methods of text objects in Dreamweaver, along with explanations where appropriate. A bullet (•) marks read-only properties.

Property or method	Return value and explanation
<code>nodeType</code> •	<code>Node.TEXT_NODE</code>
<code>parentNode</code> •	The parent tag.
<code>childNodes</code> •	An empty <code>NodeList</code> .
<code>data</code>	The actual text string. Entities in the text are represented as a single character (for example, the text <code>Joseph & I</code> is returned as <code>Joseph & I</code>).
<code>hasChildNodes()</code>	<code>false</code>

Properties and methods of comment objects

Each HTML comment is represented by a JavaScript object. The following table lists the properties and methods of comment objects in Dreamweaver, along with explanations where appropriate. A bullet (•) marks read-only properties.

Property or method	Return value and explanation
<code>nodeType</code> •	<code>Node.COMMENT_NODE</code>
<code>parentNode</code> •	The parent tag.
<code>childNodes</code> •	An empty <code>NodeList</code> .
<code>data</code>	The text string between the comment markers (<code><!--</code> and <code>--></code>).
<code>hasChildNodes()</code>	<code>false</code>

How JavaScript works in extensions

When Dreamweaver processes extensions, it compiles everything between `SCRIPT` tags and executes any code within `SCRIPT` tags that is not part of a function declaration (for example, initialization of global variables). It also reads in, compiles, and executes scripts in external JavaScript files specified in the `SRC` attributes of `SCRIPT` tags.

Note: If any JavaScript code in your extension files contains the string `'</SCRIPT>'`, the JavaScript interpreter reads this as an actual closing `SCRIPT` tag and reports an unterminated string literal error. To avoid this problem, break the string into pieces and concatenate them, like this: `'<' + '/SCRIPT>'`.

In command and behavior action files, Dreamweaver executes code in the `onLoad` event handler (if one appears in the `BODY` tag) when the user chooses the command or action from a menu. In object files, Dreamweaver executes code in the `onLoad` event handler on the `BODY` tag if the body of the document contains a form. Dreamweaver ignores the `onLoad` handler on the `BODY` tag in data translator, Property inspector, and floating panel files. In all extensions, Dreamweaver executes code in other event handlers (for example, `onBlur="alert('This is a required field.')"`) when the user interacts with the form fields to which they are attached.

Links (a tags, including those with JavaScript URLs such as ``), are not supported, nor is the `document.write()` statement. Plug-ins (set to play at all times) are supported in the `BODY` of extensions, but Java applets and ActiveX controls are not.

Running scripts at startup or shutdown

In Dreamweaver 4, if you place a command file in the `Configuration/Startup` folder, the command runs as Dreamweaver is starting up. Startup commands load before the `menus.xml` file, before the files in the `ThirdPartyTags` folder, and before any other commands, objects, behaviors, inspectors, floating panels, or translators. Thus, you can use startup commands to modify the `menus.xml` file or other extension files. You can also show warnings, prompt the user for information or call `dreamweaver.runCommand()`, but you cannot call a command that expects a valid DOM.

Similarly, if you place a command file in the `Configuration/Shutdown` folder, the command will run as Dreamweaver is shutting down. You can call the `dreamweaver.runCommand()` from shutdown commands, show warnings, or prompt the user for information, but you cannot stop the shutdown process.

For more information about commands, see “Commands” on page 37.

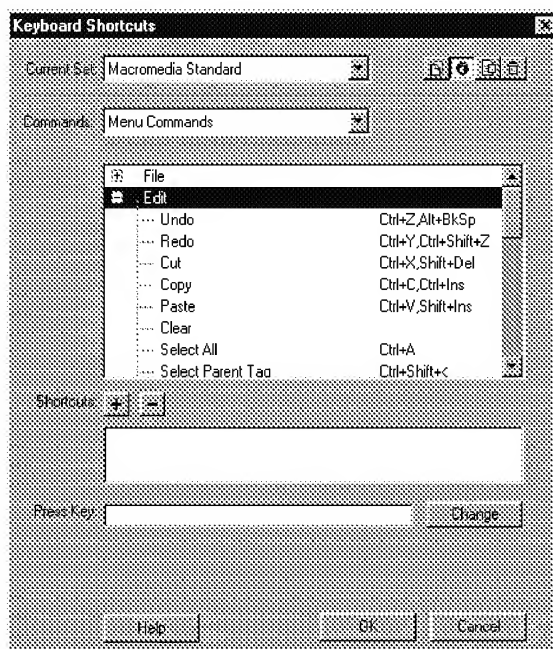
Custom JavaScript controls

Dreamweaver's Extensibility architecture provides support for the basic HTML form elements such as `select` and `input`, which can be further utilized in extensions to add new controls to the Dreamweaver UI. Two examples of this extended support are the built-in *tree control* and *color button control*.

Tree controls

The tree control displays data in a hierarchical format and allows users to expand and collapse views of the data. The data is stored in the tree in a collection of “data nodes.” Data nodes may contain other data nodes, referred to as “child nodes.” Nodes at the extreme reaches of the tree are referred to as “leaf nodes.” Leaf nodes, by definition, contain no child nodes of their own. The first node in the tree is referred to as the “root” node.

In Dreamweaver, the Keyboard Shortcuts editor uses the tree control:



Clicking on a node once selects it. In Windows, the node buttons are +/- buttons. On the Macintosh, the nodes are turn-down triangles. Nodes that contain child nodes are expandable and collapsible by clicking on the associated button next to the node.

Instantiating a tree control

A tree control uses three new tags (Dreamweaver uses the MM: prefix to distinguish the tag from standard HTML):

- MM:TREECONTROL indicates to Dreamweaver that the form element is a tree control.
- MM:TREECOLUMN specifies a column in the tree control.
- MM:TREENODE specifies one node in the tree control.

MM:TREECONTROL The MM:TREECONTROL tag is a new HTML element that the user can specify in an extension document to instantiate a tree control. The tag has several attributes, properties, events, and methods.

Although the MM:TREECONTROL tag is non-empty, it may not contain raw text. The only valid content for an MM:TREECONTROL tag is at least one or more MM:TREECOLUMN tags and zero or more MM:TREENODE tags.

The HTML attributes for an MM:TREECONTROL tag are:

Attribute Name	Description	Sample
name	Name of the tree control.	<MM:TREECONTROL name="MyTree"> </MM:TREECONTROL>
size	Number of rows to size the control for. (optional) Default is 5 rows.	<MM:TREECONTROL size="6"> </MM:TREECONTROL>
multiple	Allows a tree to have multiple selections. Default is single-selection. (optional)	<MM:TREECONTROL MULTIPLE> </MM:TREECONTROL>
style	Style definition for height and width of tree control. If specified, takes precedence over SIZE attribute. (optional)	<MM:TREECONTROL style="width:100px;height:200px" > </MM:TREECONTROL>
noheaders	Indicates that the tree column headers won't be displayed if this attribute is present.	<MM:TREECONTROL noheaders"> </MM:TREECONTROL>

MM:TREECOLUMN The MM:TREECOLUMN tag specifies a column in the tree control. You must specify at least one MM:TREECOLUMN tag.

The syntax for an MM:TREECOLUMN tag is:

Attribute Name	Description	Sample
name	Name of the column.	<MM:TREECOLUMN name="Column1">
value	String to appear in column header.	<MM:TREECOLUMN value="Files">
width	Width of the column, in pixels. Percentage not supported. Default is 100.	<MM:TREECOLUMN width="60">
align	Alignment of column text - 'left', 'right', or 'center'. Default is left.	<MM:TREECOLUMN align="center">
state	State of the column- "visible" or "hidden".	<MM:TREECOLUMN state="visible">

TREECOLUMN tags should appear at the top of the TreeControl declaration, such as:

```
<MM:TREECONTROL name="tree1">  
<MM:TREECOLUMN name="Column1" width="100" state="visible">  
<MM:TREECOLUMN name="Column2" width="80" state="visible">  
...  
</MM:TREECONTROL>
```

Note: TREECOLUMN tags can technically appear anywhere in the TREECONTROL tag, but should be placed at the top for readability.

MM:TREENODE The MM:TREENODE tag specifies an individual node within an MM:TREECONTROL. It is a non-empty tag and may contain only other MM:TREENODE tags.

The MM:TREENODE tag may not contain raw text, even though it is non-empty. The only valid content for an MM:TREENODE tag is zero or more MM:TREENODE tags.

The MM: TREENODE attributes are:

Attribute Name	Description	Sample
name	Name of the TREENODE tag.	<MM:TREENODE name="node1"> </MM:TREENODE>
value	Contains the content for the given node. For more than one column, this will be a pipe-delimited string. To specify an empty column, place a single space character before the pipe.	<MM:TREENODE value="Lawnmower 129.95"> </MM:TREENODE> // EMPTY COL EXAMPLE: <MM:TREENODE value="Data "> </MM:TREENODE>
state	State of the node - "expanded" or "collapsed".	<MM:TREENODE state="expanded"> </MM:TREENODE>
selected	True if the node is currently selected. You can select multiple nodes by setting this attribute on more than one tree node, if the tree has a MULTIPLE attribute.	<MM:TREENODE selected> </MM:TREENODE>
icon	Integer index of built-in icon to use, starting with 0. The built-in icons are: 0 = no icon 1 = DW document icon 2 = multi-document icon 3 = folder 4 = image document 5 = green check 6 = red X 7 = question mark 8 = note icon 9 = warning icon 10 = stop icon	<MM:TREENODE icon='1'> </MM:TREENODE>

A typical set of tree control code might look like:

```
<MM:TREECONTROL name="CtrlName" [size=N]
[style="[width:#px];[height:#px]"]>
<MM:TREECOLUMN name="Column1" value="Items">
<MM:TREENODE value="Item1" selected></MM:TREENODE>
<MM:TREENODE value="Item2|Item3" expanded>
<MM:TREENODE value="Item4|Item5"></MM:TREENODE>
</MM:TREENODE>
</MM:TREECONTROL>
```

Manipulating tree control content

TREECONTROLS and TREENODES are implemented as HTML tags, and are thus parsed by Dreamweaver and stored in the document tree. These tags can then be manipulated just like any other document node by calling DOM functions and using DOM properties already built into Dreamweaver.

Adding nodes Currently, Dreamweaver does not have a DOM function for creating new nodes within the document structure. Dreamweaver does support two properties, innerHTML and outerHTML, that can be used to simulate this function.

For example, to add a child node to an existing empty parent node, you would set:

```
parentNode.innerHTML = "<MM:TREENODE name='X' value='myNewNode'-'>expanded</MM:TREENODE>"
```

Deleting nodes Currently, Dreamweaver does not support a DOM function for deleting a node from the document structure. This behavior can be simulated using the innerHTML and outerHTML properties.

For example, to delete all child nodes of a given parent node, you would set:

```
parentNode.innerHTML = ""
```

Moving nodes Dreamweaver does not currently provide a DOM function for moving one node to another location within the document. This action can be simulated using a combination of adding and deleting nodes (using the innerHTML and outerHTML properties).

Modifying nodes Nodes can be directly modified using the provided Dreamweaver DOM functions.

For example, to set the “expanded” attribute of a tree node, a scripter can simply use the DOM function `NODE.setAttribute('expanded', 'expanded')` to expand the node. All other attributes can be controlled the same way.

Traversing nodes Dreamweaver does not have built in API functions for traversing, or iterating, through nodes. This behavior can be simulated using the `parentNode`, `childNodes`, and `hasChildNodes()` properties and methods.

For example, to retrieve the parent node of a node, you would use:

```
parent = node.parentNode;
```

Getting node data Node data can be retrieved directly from the node using the provided Dreamweaver DOM node functions and properties such as `getAttribute()`, `innerHTML`, and `outerHTML`.

For example, to see if a node was selected, you could use:

```
bSelected = (node.getAttribute('selected') != null)
```


A Color Button Control for extensions

A Color Button Control lets you add a color picker interface to your extensions. You instantiate a color button using the existing HTML form input tag with the `type` attribute of the control set to `mmcolorbutton`, and then provide the `name` and `value` attributes accordingly. The `name` attribute is the name of the color button. The `value` attribute is the current color value, represented as either a hexadecimal string or as a color name.

Here are some examples of the `mmcolorbutton` tag:

```
<input type="mmcolorbutton" name="colorbutton" value="#FF0000">  
<input type="mmcolorbutton" name="colorbutton" value="teal">
```

A color button has one event, `onChange`, which is triggered when the color is changed.

CHAPTER 3

Objects

Objects are designed to insert a specific string of code into the user's document. An object appears in a category on the Objects panel and in the Insert menu once its Object file is stored in a subfolder within the Configuration/Objects folder inside the Dreamweaver application folder.

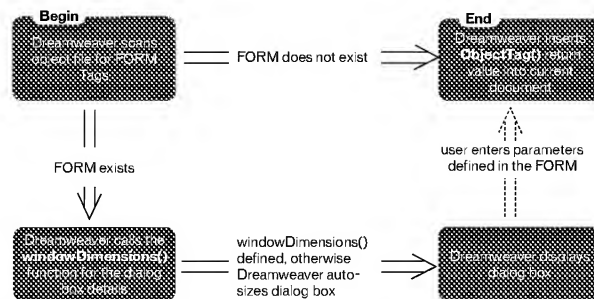
Objects have two components: the Object file that defines what is inserted in your document; and the 18 by 18 pixel GIF image that appears in the Objects panel.

Objects are HTML files. The BODY of an Object file can contain an HTML form that accepts parameters for the object (for example, the number of rows and columns in the table to be inserted). The HEAD of an Object file contains JavaScript functions that process form input from the BODY and control what is added to the user's document.

Note: The simplest objects contain only the HTML to be inserted, with no BODY and HEAD tag. See "Customizing Dreamweaver" in *Using Dreamweaver* for more information.

How object files work

When a user selects an object by clicking an icon in the Objects panel or by choosing an item in the Insert menu, the following chain of events occurs:



Object Process diagram

- 1 The Object file is scanned for a FORM tag. If a form exists—and if the Show Dialog When Inserting Objects option is selected in the General preferences—Dreamweaver calls the `windowDimensions()` function, if defined, to determine the size of the dialog box in which to display the form. If no form exists in the Object file, Dreamweaver does not display a dialog box, and step 2 is skipped.
- 2 If Dreamweaver displayed a dialog box in step 1, the user enters parameters for the object (such as the number of rows and columns in a table) in the dialog box and clicks OK.
- 3 The `objectTag()` function is called, and its return value is inserted into the document after the current selection (it does not replace the current selection).

The object API

The custom functions in the object API are not required. The functions in the object API differ from the functions in the main JavaScript API in three ways:

- They are not methods of the `dreamweaver`, `dom`, or `site` object.
- They are significant only in the context of Object files. That is, Dreamweaver automatically calls the `objectTag()` function if it is defined in an Object file, whereas a function named `objectTag()` acts like a user-defined function in any other extension file—you have to call it explicitly.
- You are responsible for writing the body of each function and returning a value, if required. This is the opposite of how the functions work in the main API: those you call and pass arguments to, and Dreamweaver generates return values, if any. Here Dreamweaver calls the functions and passes arguments to them, and you generate return values, if any.

`displayHelp()`

Description

If this function is defined, displays a Help button below the OK and Cancel buttons in the Parameters dialog box. This function is called when the user clicks the Help button.

Arguments

None.

Returns

Nothing.

Example

The following instance of `displayHelp()` opens in a browser window a file with instructions for using the object:

```
function displayHelp(){
    dreamweaver.browseDocument('http://people.netscape.com/
andreww/dreamweaver-
/objects.html');
}
```

isDomRequired()

Description

Determines whether the object requires a valid DOM to operate. If this function returns `true` or if the function is not defined, Dreamweaver assumes that the command requires a valid DOM and synchronizes the Code and Design views for the document before executing. Synchronization causes all edits in the Code view to be updated in the Design view.

Arguments

None.

Returns

`true` if a command requires a valid DOM to operate, `false` if not.

objectTag()

Description

Inserts a string of code into the user's document.

Arguments

None.

Returns

The string to be inserted.

Example

The following instance of `objectTag()` inserts an `OBJECT/EMBED` combination for a specific ActiveX control and plug-in:

```
function objectTag() {
return '\n' +
'<OBJECT CLASSID="clsid:166F100B-3A9R-11FB-8075444553540000" \n' +
'CODEBASE="http://www.mysite.com/product/cabs/-
myproduct.cab#version=1,0,0,0" \n' +
'NAME="MyProductName"> \n' +
'<PARAM NAME="SRC" VALUE=""> \n' +
'<EMBED SRC="" HEIGHT="" WIDTH="" NAME="MyProductName"> \n' +
'</OBJECT>'
}
```

windowDimensions()

Description

Sets specific dimensions for the Parameters dialog box. If this function is not defined, the window dimensions are computed automatically.

Note: Do not define this function unless you want an Options dialog box larger than 640 x 480 pixels.

Arguments

platform

The value of the argument is either "macintosh" or "windows", depending on the user's platform.

Returns

A string of the form "widthInPixels,heightInPixels".

The returned dimensions are smaller than the size of the entire dialog box because they do not include the area for the OK and Cancel buttons. If the returned dimensions do not accommodate all options, scroll bars appear.

Example

The following instance of windowDimensions() sets the dimensions of the Parameters dialog box to 648 x 520 pixels if the platform is Windows, and 660 x 580 pixels if the platform is Macintosh:

```
function windowDimensions(platform){
    var retval = ""
    if (platform == "windows"){
        retval = "648, 520";
    }else{
        retval = "660, 580";
    }
    return retval;
}
```

Adding objects to the Objects panel

Dreamweaver automatically adds any files that are inside one of the subfolders in the Configuration/Objects folder to the category associated with the subfolder. For example, a file inside the Configuration/Objects/MyObjects folder would appear on the MyObjects category of the Objects panel.

Note: Although Object files can be stored in separate folders, it's important that their file names be unique. The `dom.insertObject()` function, for example, looks for a specified file anywhere within the Objects folder without regard to subfolders. If a file called `Button.htm` exists in the Forms folder and also in the MyObjects folder, Dreamweaver cannot distinguish between them.

Each Object file has an associated 18 x 18 pixel GIF image that appears in the Objects panel. The Image file must have the same base name as the Object file (for example, `object.gif` is associated with `object.htm`) to ensure that the files remain connected.

If you create a larger object image, Dreamweaver will scale it to 18 x 18 pixels. If you do not create an image for your object, a default object icon appears in the Objects panel.

Adding objects to the Insert menu

Dreamweaver automatically adds to the bottom of the Insert menu any files that are inside one of the subfolders in the Configuration/Objects folder.

To control the position of an object in the Insert menu or any other menu, or to add an object to multiple menus, you can modify the `menus.xml` file. This file controls the entire menu structure for Dreamweaver. For more information about modifying the `menus.xml` file, see “Customizing Dreamweaver” in *Using Dreamweaver*.

Note: In Dreamweaver 2 and earlier, the items in the Insert menu were controlled by a file called `InsertMenu.htm`. This file has been superseded by `menus.xml`.

CHAPTER 4

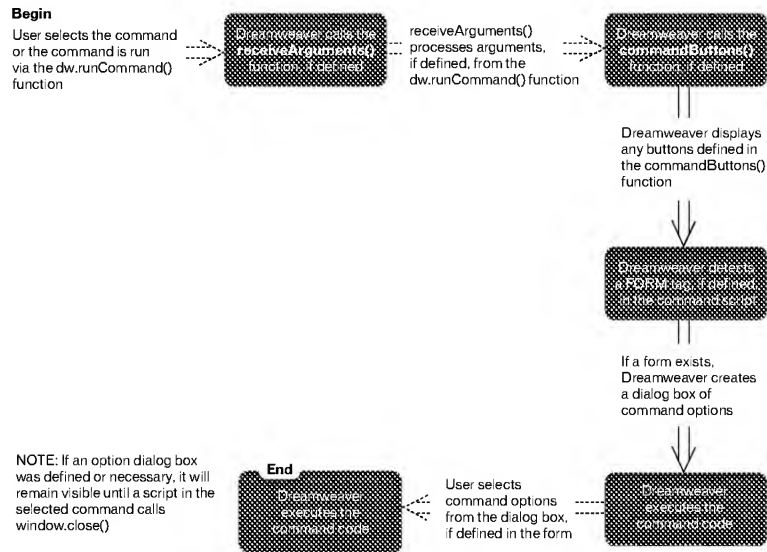
Commands

Commands can be used to perform almost any kind of edit to the user's current document, other open documents, or to any HTML document on a local drive. Commands can insert, remove, or rearrange HTML tags and attributes, comments, and text.

Commands are HTML files. The `BODY` of a Command file can contain an HTML form that accepts options for the command (for example, how a table should be sorted and by which column). The `HEAD` of a Command file contains JavaScript functions that process form input from the `BODY` and control what edits are made to the user's document.

How commands work

When a user clicks a menu that contains a command, the following chain of events occurs:



Command process and dialog box integration diagram

- 1 Dreamweaver calls the `canAcceptCommand()` function, if defined, in each Command file referenced in the menu to see whether this command is appropriate for the selection. If `canAcceptCommand()` returns `false`, the command is dimmed in the menu.
- 2 The user chooses a command from the menu.
- 3 Dreamweaver calls the `receiveArguments()` function, if defined, in the selected Command file to give the command an opportunity to process any arguments passed from the `dreamweaver.runCommand()` function.

- 4 Dreamweaver calls the `commandButtons()` function, if defined, to determine which buttons appear along the right side of the Options dialog box and what code should be executed when the user clicks the buttons.
- 5 Dreamweaver scans the Command file for a `FORM` tag. If a form exists, Dreamweaver calls the `windowDimensions()` function to determine the size of the Options dialog box containing the `BODY` elements of the file. If `windowDimensions()` is not defined, the size of the dialog box is determined automatically.
- 6 If the Command file's `BODY` tag contains an `onLoad` handler, Dreamweaver executes it (whether or not a dialog box is displayed). If no dialog box appears, the remaining steps do not occur.
- 7 The user selects options for the command. Dreamweaver executes event handlers associated with the fields as the user encounters them.
- 8 The user clicks one of the buttons defined by `commandButtons()`.
- 9 Dreamweaver executes the code associated with the button that was clicked.
- 10 The dialog box remains visible until one of the scripts in the command calls `window.close()`.

The command API

The custom functions in the command API are not required. The functions in the command API differ from the functions in the main JavaScript API in three ways:

- They are not methods of the `dreamweaver`, `dom`, or `site` object.
- They are significant only in the context of Command files. That is, Dreamweaver automatically calls the `commandButtons()` function if it is defined in a Command or Menu Command file, whereas a function named `commandButtons()` acts like a user-defined function in any other extension file—you have to call it explicitly.
- You are responsible for writing the body of each function and returning a value, if required. This is the opposite of how the functions work in the main API: those you call and pass arguments to, and Dreamweaver generates return values, if any. Here Dreamweaver calls the functions and passes arguments to them, and you generate return values, if any.

canAcceptCommand()

Description

Determines whether the command is appropriate for the current selection.

Note: Do not define `canAcceptCommand()` unless it returns `false` in at least one case. If the function is not defined, the command is assumed to be appropriate; making this assumption saves time and improves performance.

Arguments

None.

Returns

`true` if the command is allowed; `false` if it is not, dimming the command in the menu.

Example

The following instance of `canAcceptCommand()` makes the command available only when the selection is a table:

```
function canAcceptCommand(){
    var selObj=dw.getDocumentDOM.getSelectedNode();
    return (selObj.nodeType == Node.ELEMENT_NODE && ¬
        selObj.tagName=="TABLE");
}
```

commandButtons()

Description

Defines the buttons that should appear on the right side of the Options dialog box and their behavior when clicked. If this function is not defined, no buttons appear, and the BODY of the Command file expands to fill the entire dialog box.

Arguments

None.

Returns

An array containing an even number of elements. The first element is a string containing the label for the topmost button. The second element is a string of JavaScript code that defines the behavior of the topmost button when clicked. Remaining elements define additional buttons in the same manner.

Example

The following instance of `commandButtons()` defines three buttons: OK, Cancel, and Help.

```
function commandButtons(){
    return new Array("OK" , "doCommand()" , ¬
        "Cancel" , "window.close()" , "Help" , "showHelp()");
}
```

isDomRequired()

Description

Determines whether the command requires a valid DOM to operate. If this function returns `true` or if the function is not defined, Dreamweaver assumes that the command requires a valid DOM and synchronizes the Design and Code views of the document before executing. Synchronization causes all edits in the code view to be updated in the Design view.

Arguments

None.

Returns

`true` if a command requires a valid DOM to operate, `false` if not.

receiveArguments()

Description

Processes any arguments passed from a menu item or from `dw.runCommand()`, if any arguments were given to the `dw.runCommand()` function.

Arguments

{arg1}, {arg2},...{argN}

If the `arguments` attribute is defined for a `menuItem` tag, the value of that attribute is passed to the `receiveArguments()` function as one or more arguments. The `arguments` attribute is useful for distinguishing between two menu items that call the same menu command. Arguments can also be passed to a command by the `dw.runCommand()` function.

Returns

Nothing.

windowDimensions()

Description

Sets specific dimensions for the Parameters dialog box. If this function is not defined, the window dimensions are computed automatically.

Note: Do not define this function unless you want an options dialog box larger than 640 x 480 pixels.

Arguments

platform

The value of the argument is either "macintosh" or "windows", depending on the user's platform.

Returns

A string of the form "widthInPixels,heightInPixels".

The returned dimensions are smaller than the size of the entire dialog box because they do not include the area for the OK and Cancel buttons. If the returned dimensions do not accommodate all options, scroll bars appear.

Example

The following instance of windowDimensions() sets the dimensions of the Parameters dialog box to 648 x 520 pixels:

```
function windowDimensions(){  
    return "648,520";  
}
```

A simple command example

The following command converts the selected text to all lowercase characters. The command is very simple; it does not display a dialog box, so the commandButtons() function is not defined.

```
<HTML>  
<HEAD>  
<TITLE>Make Lower Case</TITLE>  
<SCRIPT LANGUAGE="javascript">  
  
function canAcceptCommand(){  
    // Get the DOM of the current document  
    var theDOM = dw.getDocumentDOM();  
    // Get the offsets of the selection  
    var theSel = theDOM.getSelection();  
    // Get the selected node  
    var theSelNode = theDOM.getSelectedNode();  
    // Get the children of the selected node  
    var theChildren = theSelNode.childNodes;
```

```

    // If the selection is not an insertion point, and
    // either the selection or its first child is a
    // text node, return true.
    return (theSel[0] != theSel[1] && (theSelNode.nodeType == Node.TEXT_NODE || theChildren[0].nodeType == Node.TEXT_NODE));
}

function changeToLowerCase() {
    // Get the DOM again
    var theDOM = dw.getDocumentDOM();
    // Get the offsets of the selection
    var theSel = theDOM.getSelection();

    // Get the outerHTML of the HTML tag (the
    // entire contents of the document)
    var theDocEl = theDOM.documentElement;
    var theWholeDoc = theDocEl.outerHTML;

    // Extract the selection
    var selText = theWholeDoc.substring(theSel[0],theSel[1]);

    // Re-insert the modified selection into the document
    theDocEl.outerHTML = theWholeDoc.substring(0,theSel[0]) + selText.toLowerCase() + theWholeDoc.substring(theSel[1]);

    // Set the selection back to where it was when you
    // started
    theDOM.setSelection(theSel[0],theSel[1]);
}

</SCRIPT>
</HEAD>

<BODY onLoad="changeToLowerCase()">
    • <!-- The function that does all the work in this command is -->
    • called from the onLoad handler on the BODY tag. There is no -->
    • form in the BODY, so no dialog box appears. -->

</BODY>
</HTML>

```

Adding commands to the Commands menu

Dreamweaver automatically adds to the bottom of the Commands menu any files that are inside the Configuration/Commands folder. To prevent a command from appearing in the Commands menu, put the following comment on the first line of the file:

```
<!-- MENU-LOCATION=NONE -->
```

In Dreamweaver 1 and 2, if you wanted to control the position or text associated with a command, you could add it explicitly to the CommandMenu.htm file. In Dreamweaver 3, this file was superseded by the menus.xml file, which controls the entire menu structure for Dreamweaver. For more information about modifying the menus.xml file, see “Customizing Dreamweaver” in *Using Dreamweaver*.

CHAPTER 5

Menu Commands

Dreamweaver 3 introduced a new type of command that made menus more flexible and dynamic menus possible. Like regular commands, menu commands can be used to perform almost any kind of edit to the current document, other open documents, or any HTML document on a local drive. The menu commands API expands the regular command API to accomplish several tasks related to displaying and calling the command from the menu system.

Note: Because menu commands are directly related to the menu system in Dreamweaver, you should read “Customizing Dreamweaver,” in *Using Dreamweaver* before continuing.

Menu commands are HTML files. The BODY of a Menu Command file can contain an HTML form that accepts options for the command (for example, how a table should be sorted and by which column). The HEAD of a Menu Command file contains JavaScript functions that process form input from the BODY and control what edits are made to the user’s document.

Menu commands are stored in the Configuration/Menus folder inside the Dreamweaver application folder.

Note: If you are adding your own menu commands to Dreamweaver, add them at the top level of the Menus folder or create your own subfolder. The MM folder is reserved for the menu commands that ship with Dreamweaver.

How menu commands work

When the user clicks a menu that contains a menu command, the following chain of events occurs:

- 1 If any `menuItem` tag in the menu contains the `dynamic` attribute, Dreamweaver calls the `getDynamicContent()` function in the associated Menu Command file to populate the menu.
- 2 Dreamweaver calls the `canAcceptCommand()` function in each Menu Command file referenced in the menu to check whether the command is appropriate for the selection. If `canAcceptCommand()` returns `false`, the menu item is dimmed.

If `canAcceptCommand()` returns `true` or is not defined, Dreamweaver calls the `isCommandChecked()` function to determine whether to display a check mark next to the menu item. If `isCommandChecked()` is not defined, no check mark is displayed.
- 3 Dreamweaver calls the `setMenuText()` function to determine the text that should appear in the menu. If `setMenuText()` is not defined, Dreamweaver uses the text that is specified in the `menuItem` tag.
- 4 The user selects an item from the menu.
- 5 Dreamweaver calls the `receiveArguments()` function, if defined, in the selected Menu Command file to let the command process any arguments passed from the menu item.
- 6 Dreamweaver calls the `commandButtons()` function, if defined, to determine which buttons appear along the right side of the Options dialog box and what code should be executed when the user clicks the buttons.
- 7 Dreamweaver scans the Menu Command file for a `FORM` tag. If a form exists, Dreamweaver calls the `windowDimensions()` function to determine the size of the Options dialog box containing the `BODY` elements of the file. If `windowDimensions()` is not defined, the size of the dialog box is determined automatically.
- 8 If the Menu Command file's `BODY` tag contains an `onLoad` handler, Dreamweaver executes the code associated with the handler (whether or not a dialog box is displayed). If no dialog box appears, the remaining steps do not occur.
- 9 The user selects options in the dialog box. Dreamweaver executes event handlers associated with the fields as the user encounters them.
- 10 The user clicks one of the buttons defined by `commandButtons()`.
- 11 Dreamweaver executes the code associated with the clicked button.
- 12 The dialog box remains visible until one of the scripts in the menu command calls `window.close()`.

The menu command API

The custom functions in the menu command API are not required. The functions in the menu command API differ from the functions in the main JavaScript API in three ways:

- They are not methods of the `dreamweaver`, `dom`, or `site` object.
- They are significant only in the context of Menu Command files. That is, Dreamweaver automatically calls the `getDynamicContent()` function if it is defined in a Menu Command file, whereas a function named `getDynamicContent()` acts like a user-defined function in any other extension file—you have to call it explicitly.
- You are responsible for writing the body of each function and returning a value, if required. This is the opposite of how the functions work in the main API: those you call and pass arguments to, and Dreamweaver generates return values, if any. Here Dreamweaver calls the functions and passes arguments to them, and you generate return values, if any.

`canAcceptCommand()`

Description

Determines whether the menu item should be active or dimmed.

Arguments

{arg1}, {arg2},...{argN}

If the `arguments` attribute is defined for a `menuItem` tag, the value of that attribute is passed to the `canAcceptCommand()` function (and to the `isCommandChecked()`, `receiveArguments()`, and `setMenuText()` functions) as one or more arguments. The `arguments` attribute is useful for distinguishing between two menu items that call the same menu command.

Returns

A Boolean value indicating whether the item should be enabled.

commandButtons()

Description

Defines the buttons that should appear on the right side of the Options dialog box and their behavior when clicked. If this function is not defined, no buttons appear, and the BODY of the Menu Command file expands to fill the entire dialog box.

Arguments

None.

Returns

An array containing an even number of elements. The first element is a string containing the label for the topmost button. The second element is a string of JavaScript code that defines the behavior of the topmost button when clicked. The remaining elements define additional buttons in the same manner.

Example

The following instance of `commandButtons()` defines three buttons: OK, Cancel, and Help.

```
function commandButtons(){  
    return new Array("OK" , "doCommand()" , "Cancel" , "  
        "window.close()" , "Help" , "showHelp()");  
}
```

getDynamicContent()

Description

Retrieves the content for the dynamic portion of the menu.

Arguments

menuID

The argument is the value of the `id` attribute in the `menuItem` tag associated with the item.

Returns

An array of strings. Each string contains the name of a menu item and its unique ID, separated by a semicolon. If the function returns `null`, then no change is made to the menu.

Example

The following instance of `getDynamicContent()` returns an array of four menu items (My Menu Item 1, My Menu Item 2, and so on):

```
function getDynamicContent(){
    var stringArray= new Array();
    var i=0;
    var numItems = 4;

    for (i=0; i<numItems;i++)
        stringArray[i] = new String("My Menu Item " + i + ";id=" + i);

    return stringArray;
}
```

isCommandChecked()

Description

Determines whether to display a check mark next to the menu item.

Arguments

{arg1}, {arg2},...{argN}

If the `arguments` attribute is defined for a `menuItem` tag, the value of that attribute is passed to the `isCommandChecked()` function (and to the `canAcceptCommand()`, `receiveArguments()`, and `setMenuText()` functions) as one or more arguments. The `arguments` attribute is useful for distinguishing between two menu items that call the same menu command.

Returns

A Boolean value indicating whether a check mark should appear next to the menu item.

receiveArguments()

Description

Processes any arguments passed from a menu item or from `dw.runCommand()`, if any arguments were given to the `dw.runCommand()` function.

Arguments

{arg1}, {arg2},...{argN}

If the `arguments` attribute is defined for a `menuItem` tag, the value of that attribute is passed to the `receiveArguments()` function (and to the `canAcceptCommand()`, `isCommandChecked()`, and `setMenuText()` functions) as one or more arguments. The `arguments` attribute is useful for distinguishing between two menu items that call the same menu command.

Returns

Nothing.

setMenuText()

Description

Specifies the text that should appear in the menu.

Note: Do not use this function if you are using `getDynamicContent()`.

Arguments

{arg1}, {arg2},...{argN}

If the `arguments` attribute is defined for a `menuItem` tag, the value of that attribute is passed to the `setMenuText()` function (and to the `canAcceptCommand()`, `isCommandChecked()`, and `receiveArguments()` functions) as one or more arguments. The `arguments` attribute is useful for distinguishing between two menu items that call the same menu command.

Returns

The string that should appear in the menu.

windowDimensions()

Description

Sets specific dimensions for the Parameters dialog box. If this function is not defined, the window dimensions are computed automatically.

Note: Do not define this function unless you want an Options dialog box larger than 640 x 480 pixels.

Arguments

platform

The value of the argument is either "macintosh" or "windows", depending on the user's platform.

Returns

A string of the form "widthInPixels,heightInPixels".

The returned dimensions are smaller than the size of the entire dialog box because they do not include the area for the OK and Cancel buttons. If the returned dimensions do not accommodate all options, scroll bars appear.

Example

The following instance of `windowDimensions()` sets the dimensions of the Parameters dialog box to 648 x 520 pixels:

```
function windowDimensions(){  
    return "648,520";  
}
```

A simple menu command

The following menu command is associated with two menu items: Undo and Redo. It checks the arguments attribute of the menuitem tag and performs a `dw.undo()` or a `dw.redo()` operation depending on the value of the first (and only) argument.

```
<HTML>
<HEAD>
<!-- Copyright 1999 Macromedia, Inc. All rights reserved. -->
<TITLE>Edit Clipboard</TITLE>
<SCRIPT LANGUAGE="javascript">
function receiveArguments()
{
    if (arguments.length != 1) return;

    var whatToDo = arguments[0];
    if (whatToDo == "undo")
    {
        dw.undo();
    }
    else if (whatToDo == "redo")
    {
        dw.redo();
    }
}

function canAcceptCommand()
{
    var selarray;
    if (arguments.length != 1) return false;
    var bResult = false;

    var whatToDo = arguments[0];
    if (whatToDo == "undo")
    {
        bResult = dw.canUndo();
    }
    else if (whatToDo == "redo")
    {
        bResult = dw.canRedo();
    }
    return bResult;
}

function setMenuText()
{
    if (arguments.length != 1) return "";

    var whatToDo = arguments[0];
    if (whatToDo == "undo")
        return dw.getUndoText();
}
```

```

        else if (whatToDo == "redo")
            return dw.getRedoText();
        else return "";
    }

```

```

</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>

```

In this command, the `receiveArguments()` function both processes the arguments and executes the command, but this need not be the case. More complex menu commands might call different functions to execute the command. For example, the following code checks whether the first argument is "foo"; if it is, it calls the `doOperationX()` function and passes it the second argument. If the first argument is "bar", it calls the `doOperationY()` function and passes it the second argument. `doOperationX()` or `doOperationY()` is responsible for executing the command.

```

function receiveArguments(){
    if (arguments.length != 2) return;

    var whatToDo = arguments[0];

    if (whatToDo == "foo"){
        doOperationX(arguments[1]);
    }else if (whatToDo == "bar"){
        doOperationY(arguments[1]);
    }
}

```


A simple dynamic menu

The following menu command does two things: It generates the Preview in Browser submenu, and it launches the current file (or the selected files in the Site window) in the browser that the user chooses from the submenu.

```
<HTML>
<HEAD>
<!-- Copyright 1999 Macromedia, Inc. All rights reserved. -->
<TITLE>Preview Browsers</TITLE>
<SCRIPT LANGUAGE="javascript">
<!--
    // getDynamicContent returns the contents of a dynamically
    // generated menu.
    // returns an array of strings to be placed in the menu, with a
    // unique
    // identifier for each item separated from the menu string by a
    // semicolon.
    //
    // return null from this routine to indicate that you are not
    // adding any
    // items to the menu
    function getDynamicContent(itemID)
    {
        var browsers = null;
        var PIB = null;
        var i;
        var j=0;
        var bUpdate = dw.getMenuNeedsUpdating(itemID);

        if (bUpdate)
        {
            browsers = new Array();
            PIB = dw.getBrowserList();
            // each browser pair has the name of the browser and the path
            // that leads
            // to the application on disk. We only put the names in the
            // menus.
            for (i=0; i<PIB.length; i=i+2)
            {
                browsers[j] = new String(PIB[i]);

                if (dw.getPrimaryBrowser() == PIB[i+1])
                    browsers[j] += "\tF12";
                if (navigator.platform == "MacPPC")
                {
                    if (dw.getSecondaryBrowser() == PIB[i+1])
                        browsers[j] += "\t ?F12";
                }
            }
            else
            {
                if (dw.getSecondaryBrowser() == PIB[i+1])
```

```

        browsers[j] += "\t Ctrl+F12";
    }

    browsers[j] += ";id='"+PIB[i]+'";
    j = j+1;
}
dw.notifyMenuUpdated(itemID, "dw.getBrowserList()");
}
return browsers;
}

function canAcceptCommand()
{
    var bHaveDocument;

    if (dw.getFocus() == 'site')
        bHaveDocument = site.getSelection().length > 0;
    else
        bHaveDocument = dw.getDocumentDOM('document') != null;

    return bHaveDocument;
}

function receiveArguments()
{
    var theBrowser = arguments[0];
    if (dw.getFocus() == 'site')
        dw.browseDocument(site.getSelection(),theBrowser);
    else

dw.browseDocument(dw.getDocumentPath('document'),theBrowser);
}

// -->
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>

```

CHAPTER 6

Reports

You can use the reports API functions to create custom reports or modify the set of prewritten reports shipped with Dreamweaver.

Reports are stored in the \Configuration\Reports folder. The Reports folder has subfolders representing report categories. Each report can belong to only one category.

Each subfolder can have a file in it called _foldername.txt. If this file is present, its contents are used as the category name instead of the folder name. If this file is not present, the folder name is used. The category names are limited to 31 characters.

How reports work

- 1 Reports are accessible through the Site > Reports... menu. When chosen, this menu item displays a dialog box from which the user selects reports to run on a choice of targets.
- 2 The user selects which files to run the selected reports on using the Report On: menu. This menu contains Current Document, All Files in Current Local Site, Selected Files In Local Site, and Folder. When the Folder option is selected, a Browse button and edit field appear to let the user choose a folder.
- 3 The user can customize reports that have parameters by choosing the Settings button, then entering values for the parameters. Each report is responsible for displaying its own Settings dialog box. This dialog box is optional; not every report will require the user to set the report's parameters. If a report does not have a Settings dialog box, then the Report Settings... button is dimmed when the report is selected in the list.
- 4 After the reports are selected and their settings set, the user clicks the Run button.

Each report that defines the `beforeReporting()` function has that function called before the reporting process begins. If a report returns `false` from this function, it is removed from the report run.

- 5 Each file is passed to each report that was selected in the Reports dialog box using the `processFile()` function. If the report needs to include information about this file in the results list, it should call the `dw.results.addResultItem()` function. This process continues until all files pertaining to the user's selection have been processed, or the user clicks the Stop button in the bottom of the window. Dreamweaver displays the name of each file being processed and the number of remaining files to be processed.

Each report that defines the `endReporting()` function has that function called after the reporting process completes.

- 6 The results of each report process appear in a separate Results window.

The report API

The only required function for the report API is the `processFile()` function. All other functions are optional.

`processFile()`

Description

Called when there is a file to process. The Report command should process the file without modifying it and use the `addResultItem()` function to return information about the file. Dreamweaver automatically releases each file's DOM when finished.

Arguments

strFilePath

strFilePath is a fully qualified file path name of the file to process.

Returns

Nothing.

`beginReporting()`

Description

Called at the start of the reporting process, before any reports are actually run. If the Report command returns `false` from this function, *target* is excluded from the report run.

Arguments

target

target is a string indicating the target of the report session. It can be one of the following values: "CurrentDoc", "CurrentSite", "CurrentSiteSelection" (for the selected files in a site), or "Folder:+ the path to the folder the user selected" (for example, "Folder:c:temp").

Returns

`true` if the report has run successfully, `false` if *target* is excluded from the report run.

endReporting()

Description

Called at the end of the Report process.

Arguments

None.

Returns

Nothing.

commandButtons()

Description

Defines the buttons that should appear on the right side of the Options dialog box and their behavior when clicked. If this function is not defined, no buttons appear, and the BODY of the report file expands to fill the entire dialog box.

Arguments

None.

Returns

An array containing an even number of elements. The first element is a string containing the label for the topmost button. The second element is a string of JavaScript code that defines the behavior of the topmost button when clicked. Remaining elements define additional buttons in the same manner.

Example

The following instance of `commandButtons()` defines three buttons: OK, Cancel, and Help.

```
function commandButtons(){  
    return new Array("OK" , "doCommand()" , "Cancel" , "  
    "window.close()" , "Help" , "showHelp()");  
}
```

configureSettings()

Description

Determines whether the Report Settings button should be enabled in the Reports dialog box when this report is selected.

Arguments

None.

Returns

true if the Report Settings button should be enabled, false otherwise.

windowDimensions()

Description

Sets specific dimensions for the Parameters dialog box. If this function is not defined, the window dimensions are computed automatically.

Note: Do not define this function unless you want an Options dialog box larger than 640 x 480 pixels.

Arguments

platform

The value of the argument is either "macintosh" or "windows", depending on the user's platform.

Returns

A string of the form "widthInPixels,heightInPixels".

The returned dimensions are smaller than the size of the entire dialog box because they do not include the area for the OK and Cancel buttons. If the returned dimensions do not accommodate all options, scroll bars appear.

Example

The following instance of `windowDimensions()` sets the dimensions of the Parameters dialog box to 648 x 520 pixels:

```
function windowDimensions(){  
    return "648,520";  
}
```


CHAPTER 7

JavaScript Debugger Modules

A JavaScript Debugger Module is a special type of extensibility module, which inserts special code into a document to interface with the JavaScript Debugger. The debugger modules are located with the Dreamweaver Program Files in the Configuration/Debugger subfolder. These modules insert specific JavaScript and HTML into a working document to create a “debug version” of the document the next time the JavaScript Debugger runs. The “debug version” is simply a set of temporary replicated files for the HTML document and each external JavaScript file, all created by Dreamweaver and saved in the current working folder. The debug version of the HTML file then appears in the browser. The JavaScript inserted into the temporary files, called “instrumentation”, communicates with the Dreamweaver JavaScript Debugger as the JavaScript executes in the browser.

For information about JavaScript Debugger API Commands, see “JavaScript Debugger functions” on page 304.

How the JavaScript Debugger Module works

Dreamweaver comes with two debugger modules, one for each supported browser: Netscape and Internet Explorer. To provide support for a different browser, you would create a new debugger module, and then use `dom.instrumentDocument` and `dreamweaver.startDebugger` to debug the document in the other browser.

When you call `dom.instrumentDocument`, the specified debugger module receives callbacks as Dreamweaver parses the JavaScript in the document. So, for instance, you could create a debugger module that inserts comments or records information about the JavaScript code, instead of inserting debugging enhancements.

When `dom.instrumentDocument` is called with a specific debugger module, the following steps occur:

- 1 Dreamweaver calls `getIncludeFiles()` in the debugger module. This function returns the list of files that will be referenced from HTML instrumentation code returned from `getHeadInstrument()` and `getBodyInstrument()` (which are called later). The include files can be any type of file, such as an external JavaScript file or JavaApplet or ActiveX control. All of the files must be in the Configuration/Debugger subfolder with the Debugger Module. Dreamweaver will copy the include files to the directory that contains the file being debugged, and then later will delete the include files from that directory when Dreamweaver exits.
- 2 Next, the HTML document is scanned for script tags and event handlers. The code inside the script tag, in an external JavaScript file, or in an event handler is known as a block.

Note: An external JavaScript file is a file that is specified as the "src" attribute of a SCRIPT tag.
- 3 Dreamweaver parses script tags in the HEAD section first.
- 4 When Dreamweaver finds a script tag or event handler, it calls the debugger module's `startBlock()` function and passes in the name of the file and the line and character offsets from the beginning of the file.
- 5 Then Dreamweaver begins parsing the JavaScript code in the block.
- 6 When Dreamweaver finds a JavaScript statement, such as a variable declaration, it calls `getStepInstrument()` passing the line and character offsets and other information. The debugger module returns a string of JavaScript code that is inserted before the statement. You must take care to return valid JavaScript code. For each call to `getStepInstrument()`, Dreamweaver records the line number as a valid breakpoint line regardless of what instrumentation is returned. So, when the debugger is started with `dw.startDebugger()`, the breakpoints the user has already set will be moved to one of these valid lines.

- 7 When Dreamweaver finds a function declaration, it calls the `getFunctionStartInstrument()` to receive the instrumentation to be inserted at the beginning of the function.

Note: This is not considered a valid breakpoint line.

- 8 Dreamweaver continues parsing the function and calling `getStepInstrument()` for each statement in the function.
- 9 When Dreamweaver comes to a return statement, or the end of the function, it calls `getFunctionEndInstrument()` to receive the instrumentation to be inserted before the function returns.

Note: This is not considered a valid breakpoint line.

- 10 If Dreamweaver encounters a syntax error or warning in the JavaScript block, it calls `reportError()` or `reportWarning()`, respectively. After an error is encountered, Dreamweaver stops parsing the block. Other blocks will still be parsed.

- 11 After Dreamweaver has parsed all the script blocks in the HEAD section, it calls `getHeadInstrument()` to get the HTML instrumentation to insert in the HEAD section.

Note: This function should return HTML, not JavaScript. If the debugger module needs to insert JavaScript code in the HEAD, it must enclose it in a SCRIPT tag.

- 12 Next Dreamweaver begins processing the JavaScript blocks (SCRIPT tags and event handlers) in the BODY section of the document.

- 13 After the last block in the BODY section is instrumented, Dreamweaver calls `getBodyInstrument()` to get the HTML instrumentation to insert in the BODY section.

Note: This function should return HTML, not JavaScript.

- 14 After Dreamweaver calls `getBodyInstrument()`, there is one final call to `startBlock()` and `getStepInstrument()` for an auto-breakpoint. The instrumentation does not correspond to any user-defined SCRIPT tag, but rather is inserted in a new SCRIPT tag after the BODY instrumentation. Unlike other calls to `getStepInstrument()`, this line is not considered a valid line on which the user can set a breakpoint, but rather it is treated as a special kind of breakpoint at which the debugger always stops.

- 15 Finally, Dreamweaver calls `getOnUnloadInstrument()` to get JavaScript instrumentation to be inserted in the `onUnload` handler of the BODY tag. If the document already has an `onUnload` handler, this instrumentation is inserted after the user-defined `onUnload` code.

The JavaScript Debugger Module API

The JavaScript Debugger Module API allows you to customize the way the debug version of a document is created. You would need to create a debugger module if you wanted to make the Dreamweaver JavaScript Debugger work with a browser other than the two (Netscape and Internet Explorer) for which Dreamweaver already provides support. You could also create a debugger module for some other specialized purpose, such as counting the number of JavaScript statements used by a particular document.

Note: Currently only SCRIPT tags and event handlers are parsed for instrumentation. There are some other ways to use JavaScript in HTML documents, such as JavaScript URLs, JavaScript entities, and conditional comments. These are not currently supported.

The JavaScript Debugger Module API functions are significant only in the context of debugger module files. Specifically, Dreamweaver automatically calls the `getStepInstrument()` function if it is defined in the debugger module file. For any other extension file, a function named `getStepInstrument()` acts like a user-defined function—you have to call it explicitly.

As opposed to the way you work with the functions in the main JavaScript API, you are responsible for writing the body of each function and returning a value, if required, for the debugger modules. For the functions in the main API, you call and pass arguments, and Dreamweaver generates return values, if any. For the debugger modules, Dreamweaver calls the functions and passes arguments to them, and you generate return values, if any.

All of the JavaScript Debugger Module functions are optional. If a function is not defined, then nothing happens when Dreamweaver calls it.

`getFunctionEndInstrument()`

Availability

Dreamweaver 4.0

Description

Called after the last statement in a function declaration. If any return statements exist, then this module is called to insert instrumentation after the return value is evaluated and before the function will return strings.

Arguments

None.

Returns

A string containing the JavaScript to insert at the end of the function.

getFunctionStartInstrument()

Availability

Dreamweaver 4.0

Description

Called before the first statement in a function declaration. The `getStepInstrument()` function is also called for the statement.

Arguments

None.

Returns

A string containing the JavaScript to insert at the beginning of the function.

getBodyInstrument()

Availability

Dreamweaver 4.0

Description

This function is called exactly once after all blocks in the HEAD section have been instrumented.

Arguments

None.

Returns

A string containing HTML to insert at the top of the <BODY> section.

getHeadInstrument()

Availability

Dreamweaver 4.0

Description

This function is called exactly once after all blocks in the HEAD section are instrumented, but before the BODY section blocks are instrumented.

Arguments

None.

Returns

A string containing HTML to insert at the top of the <HEAD> section.

getIncludedFileList()

Availability

Dreamweaver 4.0

Description

Called to get a list of files that are referenced by code inserted in the head or body (from the `getHeadInstrument()` and `getBodyInstrument()` functions). These files must be located in the Configuration/Debugger directory of the Dreamweaver Program Files with the instrumentation debugger module.

Arguments

None.

Returns

An array of file names that should be copied to the directory with the instrumented file.

getOnUnloadInstrument()

Availability

Dreamweaver 4.0

Description

This function is called exactly once after `getHeadInstrument()` is called.

Arguments

None.

Returns

String containing JavaScript to insert at the end of the `onUnload` event handler of the `BODY` tag.

getStepInstrument()

Availability

Dreamweaver 4.0

Description

Called for each statement that is parsed inside a block. A call is always made to `StartBlock()` before this function is called. Dreamweaver records each line for which it calls this function as a valid breakpoint line. While the debugger is started, all breakpoints are moved to valid breakpoint lines.

Arguments

lineNumber, offset, bisInFunction

- *lineNumber* is the line number of the next statement relative to the start of the block (1-based index).
- *offset* is the offset of the first character of the next statement relative to the start of the block (0-based index).
- *bisInFunction* is a Boolean that indicates if the step is in a function definition (`true`) or in the global scope (`false`).

Returns

A string containing the JavaScript code to insert before the statement.

reportError()**Availability**

Dreamweaver 4.0

Description

Called when a syntax error is detected. The errors and warnings are not necessarily reported in order.

Arguments

fileName, errorNumber, strDesc, lineNumber, offset

- *fileName* is the name of the file.
- *errorNumber* is the numeric identifier of the error that occurred.
- *strDesc* is the description of the error.
- *lineNumber* is the line number in which the error occurred, relative to the start of the block.
- *offset* is the offset of the character at which the error occurred, relative to the start of the block.

Returns

Nothing.

reportWarning()

Availability

Dreamweaver 4.0

Description

Called when a warning is detected in the file.

Arguments

fileName, errorNumber, strDesc, lineNumber, offset

- *fileName* is the name of the file.
- *errorNumber* is the numeric identifier of the warning that occurred.
- *strDesc* is the description of the warning.
- *lineNumber* is the line number in which the warning occurred, relative to the start of the block.
- *offset* is the offset of the character at which the warning occurred, relative to the start of the block.

Returns

Nothing.

startBlock()

Availability

Dreamweaver 4.0

Description

Indicates the beginning of a new block of JavaScript code. The block can be a script tag, event handler, or external .js file.

Arguments

fileName, lineNumber, offset

- *fileName* is the name of the HTML document or .js file containing the block. The location is specified by a relative path to the source HTML document.
- *lineNumber* is the line number in the HTML document or .js file in which the block begins (1-based index).
- *offset* is the offset of the first character of JavaScript code from the beginning of the file (0-based index).

Returns

Nothing.

CHAPTER 8

Property Inspectors

The Property inspector is perhaps the most familiar floating panel in the Dreamweaver interface. It is indispensable for defining, reviewing, and changing the name, size, appearance, and other attributes of the selection, as well as for launching internal and external editors for the selected element.

Dreamweaver has several built-in interfaces for the Property inspector that let you set properties for many standard HTML tags. (These built-in inspectors are part of the core Dreamweaver code; for this reason, you will not find corresponding Property Inspector files for them in the Configuration folder.) With custom Property Inspector files, you can override these built-in interfaces or create new ones to inspect custom tags.

Custom Property Inspector files are HTML files that reside in the Configuration/Inspectors folder inside the Dreamweaver application folder. The first line of a Property Inspector file (the line above the opening HTML tag) must be a comment in the following format:

```
<!--  
tag: tagNameOrKeyword,priority:1to10,selection:exactOrWithin,hline  
,vline-->
```

where:

- *tagNameOrKeyword* is the tag to be inspected or one of the following keywords: **COMMENT** (for comments), **LOCKED** (for locked regions), or **ASP** (for ASP tags).
- *1to10* is the priority of the Property Inspector file: 1 indicates that this inspector should be used only when no others can inspect the selection; 10 indicates that this inspector takes precedence over all others that can inspect the selection.
- *exactOrWithin* indicates whether the selection can be within the tag (within) or must exactly contain the tag (exact).
- *hline* (optional) indicates that a horizontal gray line should appear between the upper and lower halves of the inspector in expanded mode.
- *vline* (optional) indicates that a vertical gray line should appear between the tag name field and the rest of the properties in the inspector (see the image Property inspector for an example).

The following comment would be appropriate for an inspector that is designed to inspect the HAPPY tag:

```
<!-- tag:HAPPY, priority:8,selection:exact,hline,vline -->
```

The BODY of a Property Inspector file contains an HTML form. Instead of displaying the form contents in a dialog box, however, Dreamweaver uses the form to define the input areas and layout of the inspector.

How Property Inspector files work

At startup, Dreamweaver reads the first line of each .htm and .html file in the Configuration/Inspectors folder, looking for the comment string that defines the type, priority, and selection type of a Property inspector. Files that do not have this comment as their first line are ignored.

When the user makes a selection in Dreamweaver or moves the insertion point to a different location, the following chain of events occurs:

- 1 Dreamweaver looks for any inspectors that have a selection type of `within`.
- 2 If there are any `within` inspectors, Dreamweaver searches up the document tree from the currently selected tag to check whether there are inspectors for any tags that surround the selection. If—and only if—there are no `within` inspectors, Dreamweaver looks for any inspectors that have a selection type of `exact`.
- 3 For the first tag found that has one or more inspectors, Dreamweaver calls each inspector's `canInspectSelection()` function. If this function returns `false`, Dreamweaver no longer considers the inspector a candidate for inspecting the selection.
- 4 If more than one potential inspector remains after calling `canInspectSelection()`, Dreamweaver sorts the remaining inspectors by priority.
- 5 If more than one potential inspector shares the same priority, Dreamweaver chooses an inspector alphabetically by name.
- 6 The chosen inspector appears in the Property Inspector floating panel. If the Property Inspector file defines the `displayHelp()` function, a small ? icon is displayed in the upper right corner of the inspector.
- 7 Dreamweaver calls the `inspectSelection()` function to gather information about the current selection and populate the inspector's fields.
- 8 Event handlers attached to the fields in the Property inspector interface execute as the user encounters them. (For example, you may have an `onBlur` event that calls `setAttribute()` to set an attribute to the value just entered by the user.)

The Property inspector API

Two of the Property inspector API functions (`canInspectSelection()` and `inspectSelection()`) are required. The Property inspector API functions differ from the functions in the main JavaScript API in three ways:

- They are not methods of the `dreamweaver`, `dom`, or `site` object.
- They are significant only in the context of Property Inspector files. That is, Dreamweaver automatically calls the `canInspectSelection()` function in a Property Inspector file, whereas a function named `canInspectSelection()` acts like a user-defined function in any other extension file—you have to call it explicitly.
- You are responsible for writing the body of each function and returning a value, if required. This is the opposite of how the functions work in the main API: those you call and pass arguments to, and Dreamweaver generates return values, if any. Here Dreamweaver calls the functions and passes arguments to them, and you generate return values, if any.

`canInspectSelection()`

Description

Determines whether the Property inspector is appropriate for the current selection.

Arguments

None.

Use `dom.getSelectedNode()` to get the current selection as a JavaScript object.

Returns

`true` if the inspector can inspect the current selection; `false` if it cannot.

Example

The following instance of `canInspectSelection()` returns `true` if the selection contains the `CLASSID` attribute and the value of that attribute is `"clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"` (the class ID for Flash Player):

```
function canInspectSelection(){
    var theDOM = dw.getDocumentDOM();
    var theObj = theDOM.getSelectedNode();
    return (theObj.nodeType == Node.ELEMENT_NODE && ¬
        theObj.hasAttribute("classid") && ¬
        theObj.getAttribute("classid").toLowerCase() == ¬
        "clsid:D27CDB6E-AE6D-11cf-96B8-444553540000");
}
```

displayHelp()

Description

If this function is defined, a ? icon appears in the upper right corner of the Property inspector. This function is called when the user clicks the icon.

Arguments

None.

Returns

Nothing.

Example

The following instance of `displayHelp()` opens a file in a browser window that explains the fields in the Property inspector:

```
function displayHelp(){
    dreamweaver.browseDocument('http://www.hooha.com/dw/↵
    inspectors/inspHelp.html');
}
```

inspectSelection()

Description

Refreshes the contents of the user input fields based on the attributes of the current selection.

Arguments

maxOrMin

The argument is either `max` or `min`, depending on whether the Property inspector is in its expanded or contracted state.

Returns

Nothing.

Example

The following instance of `inspectSelection()` gets the value of the `CONTENT` attribute and uses it to populate a form field called `keywords`:

```
function inspectSelection(){
    var dom = dreamweaver.getDocumentDOM();
    var theObj = dom.getSelectedNode();
    document.forms[0].keywords.value = ↵
    theObj.getAttribute("content");
}
```

A simple Property inspector example

The following Property inspector inspects a fictional tag called INTJ. The INTJ tag is empty (that is, it has no closing tag), so its selection type is exact. As long as the selection is exactly an INTJ tag, the Property inspector should show up—so the `canInspectSelection()` function returns true every time. To have a different inspector appear depending on the value of the INTJ tag's TYPE attribute, for example, the `canInspectSelection()` function must check the value of the TYPE attribute to determine which Property inspector is the right one. (This is how the keywords and description Property inspectors work, because “keywords” and “description” are not tags but values of the META tag's NAME attribute.)

```
<!-- tag:INTJ,priority:5,selection:exact,vline,hline -->
<HTML>
<HEAD>
<TITLE>Interjection Inspector</TITLE>
<SCRIPT LANGUAGE="JavaScript">

function canInspectSelection(){
    return true;
}

function inspectSelection(){
    // Get the DOM of the current document
    var theDOM = dw.getDocumentDOM();
    // Get the selected node
    var theObj = theDOM.getSelectedNode();

    // Get the value of the TYPE attribute on the INTJ tag
    var theType = theObj.getAttribute('type');
    // Initialize a variable called typeIndex to -1. This will be
    // used to store the menu index that corresponds to
    // the value of the TYPE attribute
    var typeIndex = -1;

    // If there was a TYPE attribute
    if (theType){
        // If the value of TYPE is "jeepers", set typeIndex to 0
        if (theType.toLowerCase() == "jeepers"){
            typeIndex = 0;
        }
        // If the value of TYPE is "jinkies", set typeIndex to 1
        else if (theType.toLowerCase() == "jinkies"){
            typeIndex = 1;
        }
        // If the value of TYPE is "zoinks", set typeIndex to 2
        else if (theType.toLowerCase() == "zoinks"){
            typeIndex = 2;
        }
    }

    // If the value of the TYPE attribute was "jeepers",
    // "jinkies", or "zoinks", choose the corresponding
```

```

        // option from the pop-up menu in the interface
        if (typeIndex != -1){
            document.topLayer.document.topLayerForm.intType.↵
                selectedIndex = typeIndex;
        }
    }

function setInterjectionTag(){
    // Get the DOM of the current document
    var theDOM = dw.getDocumentDOM();
    // Get the selected node
    var theObj = theDOM.getSelectedNode();

    // Get the index of the selected option in the pop-up menu
    // in the interface
    var typeIndex = document.topLayer.document.↵
        topLayerForm.intType.selectedIndex;
    // Get the value of the selected option in the pop-up menu
    // in the interface
    var theType = document.topLayer.document.topLayerForm.↵
        intType.options[typeIndex].value;

    // Set the value of the TYPE attribute to theType
    theObj.setAttribute('type',theType);
}

</SCRIPT>
</HEAD>

<BODY>
<SPAN ID="image" STYLE="position:absolute; width:23px;
height:17px; z-index:16; left: 3px; top: 2px">
<IMG SRC="interjection.gif" WIDTH="36" HEIGHT="36"
NAME="interjectionImage">
</SPAN>
<SPAN ID="label" STYLE="position:absolute; width:23px;
height:17px; z-index:16; left: 44px; top: 5px">Interjection</SPAN>

<!-- If your form fields are in different layers, you must
create a separate form inside each layer and reference it as
shown in the inspectSelection() and setInterjectionTag()
functions above. -->

<SPAN ID="topLayer" STYLE="position:absolute; z-index:1;
left: 125px; top: 3px; width: 431px; height: 32px">
<FORM NAME="topLayerForm">
<TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0">
<TR>
<TD VALIGN="baseline" ALIGN="right">Type:</TD>
<TD VALIGN="baseline" ALIGN="right">
<SELECT NAME="intType" STYLE="width:86"
onChange="setInterjectionTag()">

```

```
<OPTION VALUE="jeepers">Jeepers</OPTION>
<OPTION VALUE="jinkies">Jinkies</OPTION>
<OPTION VALUE="zoinks">Zoinks</OPTION>
</SELECT>
</TR>
</TABLE>
</FORM>
</SPAN>

</BODY>
</HTML>
```


CHAPTER 9

Floating Panels

.....

You can create any kind of floating panel or inspector without the size and layout limitations of Property inspectors.

A custom Property inspector should still be your first choice for setting the properties of the current selection. However, custom floating panels offer more room and flexibility for displaying information about the entire document or multiple selections.

Dreamweaver already has several built-in floating panels that are accessible from the Window menu; you can add your own panels to this menu using the extensible menus feature. (These built-in panels are part of the core Dreamweaver code; for this reason, you will not find corresponding Floating Panel files for them in the Configuration folder.) For more information on adding items to the menu system, see “Customizing Dreamweaver,” in *Using Dreamweaver*.

Custom Floating Panel files are HTML files that reside in the Configuration/Floaters folder inside the Dreamweaver application folder. The BODY of a Floating Panel file contains an HTML form; event handlers attached to form elements may call JavaScript code that performs arbitrary edits to the current document.

How Floating Panel files work

Custom floating panels can be moved, resized, and tabbed together just like the floating panels that are built into Dreamweaver. Custom floating panels differ from built-in floating panels in the following ways:

- It is not possible to display an icon in the tab of a custom floating panel; the tab always shows the contents of the floating panel's `TITLE` tag.
- Custom floating panels display in the default gray. Setting the `BGCOLOR` attribute in the `BODY` tag has no effect.
- All custom floating panels either appear always on top of the Document window or float behind it when inactive, depending on the setting for All Other Floaters in the Floating panels preferences.

Floating Panel files also differ somewhat from other extensions. Unlike other extension files, Dreamweaver does not load Floating Panel files into memory at startup unless the floating panels were visible when Dreamweaver last shut down. If the floating panels were not visible when Dreamweaver last shut down, the files that define them are loaded only when referenced from one of the following functions: `dreamweaver.getFloaterVisibility()`, `dreamweaver.setFloaterVisibility()`, or `dreamweaver.toggleFloater()`.

When one of the files inside the Configuration folder calls

```
dw.getFloaterVisibility(floaterName),  
dw.setFloaterVisibility(floaterName), or  
dw.toggleFloater(floaterName), the following chain of events occurs:
```

- 1 If *floaterName* is not one of the reserved floating panel names, Dreamweaver searches the Configuration/Floaters folder for a file called *floaterName*.htm. (For a complete list of reserved floating panel names, see “`dreamweaver.getFloaterVisibility()`” on page 450.) If *floaterName*.htm is not found, Dreamweaver searches for *floaterName*.html. If no file is found, nothing further happens.
- 2 If the Floating Panel file is being loaded for the first time, the `initialPosition()` function is called, if defined, to determine the floating panel's default position on the screen, and the `initialTabs()` function is called, if defined, to determine the floating panel's default tab grouping.
- 3 The `selectionChanged()` and `documentEdited()` functions are called on the assumption that changes probably occurred while the floating panel was hidden.

- 4 When the floating panel is visible, the following things happen:
 - When the selection changes, the `selectionChanged()` function is called, if defined.
 - When the user makes changes to the document, the `documentEdited()` function is called, if defined.
 - Event handlers attached to the fields in the floating panel interface execute as the user encounters them. (For example, a button with an `onClick` event handler that calls `dw.getDocumentDOM().body.innerHTML=''` would remove everything between the opening and closing `BODY` tags in the document when clicked.)
- 5 When the user quits Dreamweaver, the current visibility, position, and tab grouping of the floating panel are saved. The next time Dreamweaver starts up, it loads the Floating Panel files for any floating panels that were visible at the last shutdown and displays the floating panels in their last position and tab grouping.

The floating panel API

The custom functions in the floating panel API are all optional. These functions differ from the functions in the main JavaScript API in three ways:

- They are not methods of the `dreamweaver`, `dom`, or `site` object.
- They are significant only in the context of Floating Panel files. That is, Dreamweaver automatically calls the `documentEdited()` function if it is defined in a Floating Panel file, whereas a function named `documentEdited()` acts like a user-defined function in any other extension file—you have to call it explicitly.
- You are responsible for writing the body of each function and returning a value, if required. This is the opposite of how the functions work in the main API: those you call and pass arguments to, and Dreamweaver generates return values, if any. Here Dreamweaver calls the functions and passes arguments to them, and you generate return values, if any.

displayHelp()

Description

If this function is defined, a Help button appears below the OK and Cancel buttons in your dialog box. This function is called when the user clicks the Help button.

Arguments

None.

Returns

Nothing.

Example

The following instance of `displayHelp()` opens in a browser window a file with instructions:

```
function displayHelp(){
    dreamweaver.browseDocument('http://www.hotwired.com/~
    webmonkey/javascript? /code_library/wm_pos2_elmnt_dw/~
    ?tw=javascript');
}
```

documentEdited()

Description

Called when the floating panel becomes visible and after the current series of edits is complete—that is, multiple edits may occur before this function is called. This function should be defined only if the floating panel must track edits to the document.

Note: Define `documentEdited()` only if you absolutely require it because its existence impacts performance.

Arguments

None.

Returns

Nothing.

Example

The following instance of `documentEdited()` scans the document for layers and updates a text field that displays the number of layers in the document:

```
function documentEdited(){
    /* create a list of all the layers in the document */
    var theDOM = dw.getDocumentDOM();
    var layersInDoc = theDOM.getElementsByTagName("layer");
    var layerCount = layersInDoc.length;

    /* update the numOfLayers field with the new layer count */
    document.theForm.numOfLayers.value = layerCount;
}
```

initialPosition()

Description

Determines the initial position of the floating panel the first time it is called. If this function is not defined, the default position is the center of the screen.

Arguments

platform

Possible values for *platform* are "Mac" and "Win".

Returns

A string of the form "leftPosInPixels,topPosInPixels".

Example

The following instance of `initialPosition()` specifies that the first time the floating panel appears, it should be 420 pixels from the left and 20 pixels from the top in Windows, and 390 pixels from the left side of the screen and 20 pixels from the top of the screen on the Macintosh:

```
function initialPosition(platform){
    var initPos = "420,20";
    if (platform == "macintosh"){
        initPos = "390,20";
    }
    return initPos;
}
```

`initialTabs()`

Description

Determines which other floating panels are tabbed together with this one the first time the floating panel appears. If any listed floating panel has appeared previously, it is not included in the tab group. Thus, to ensure that two custom floating panels are tabbed together, each should reference the other in each `initialTabs()` function.

Arguments

None.

Returns

A string of the form "floaterName1,floaterName2,...floaterNameN".

Example

The following instance of `initialTabs()` specifies that the first time the floating panel appears, it should be tabbed together with the `scriptEditor` floating panel:

```
function initialTabs(){  
    return "scriptEditor";  
}
```

`isAvailableInCodeView()`

Description

Defined by a floating panel to determine whether the floating panel should be enabled when focus is in the Code view. If this function is not defined, the floating panel is disabled in the Code view.

Arguments

None.

Returns

A Boolean value indicating whether the floating panel should be enabled.

`selectionChanged()`

Description

Called when the floating panel becomes visible and when the selection changes (when focus switches to a new document or when the insertion pointer moves to a new location in the current document). This function should be defined only if the floating panel must track the selection.

Note: Define `selectionChanged()` only if you absolutely require it because its existence impacts performance.

Arguments

None.

Returns

Nothing.

Example

The following instance of `selectionChanged()` shows a different panel (layer) in the floating panel depending on whether the selection is a script marker or something else:

```
function selectionChanged(){
    /* get the selected node */
    var theDOM = dw.getDocumentDOM();
    var theNode = dw.getSelectedNode();

    /* check to see if the node is a script marker */
    if (theNode.nodeType == Node.ELEMENT_NODE && theNode.tagName == "SCRIPT"){
        document.layers['blanklayer'].visibility = 'hidden';
        document.layers['scriptlayer'].visibility = 'visible';
    }else{
        document.layers['scriptlayer'].visibility = 'hidden';
        document.layers['blanklayer'].visibility = 'visible';
    }
}
```

About performance

Declaring the `selectionChanged()` or `documentEdited()` function in your custom floating panels risks adversely impacting Dreamweaver performance. Consider that `documentEdited()` is called after every keystroke, and `selectionChanged()` is called after every press of an arrow key if Dreamweaver has been idle for more than one-tenth of a second. It's important to test your floating panel against many different scenarios, using large documents (100K or more of HTML) whenever possible.

To help you avoid performance penalties, `setTimeout()` was implemented as a global method in Dreamweaver 3. As in the browsers, `setTimeout()` takes two arguments: the JavaScript to be called and the amount of time in milliseconds to wait before calling it.

The `setTimeout()` method lets you build pauses into your processing during which the user can continue interacting with the application. You must build in these pauses explicitly because the screen freezes while scripts are processing, preventing the user from performing further edits (and you from updating the interface or the floating panel).

The following code is from a floating panel that displays information about every layer in the document. It uses `setTimeout()` to pause for half a second after processing each layer:

```
/* create a flag that specifies whether an edit is being -
   processed, and set it to false. */
document.running = false;

/* this function called when document is edited */
function documentEdited(){
    /* create a list of all the layers to be processed */
    var dom = dw.getDocumentDOM();
    document.layers = dom.getElementsByTagName("layer");
    document.numLayers = document.layers.length;
    document.numProcessed = 0;

    /* set a timer to call processLayer(); if we didn't get
       * to finish processing the previous edit, then the timer
       * is already set. */
    if (document.running = false){
        setTimeout("processLayer()", 500);
    }

    /* set the processing flag to true */
    document.running = true;
}

/* process one layer */
function processLayer(){
    /* display information for the next unprocessed layer.
       displayLayer() is a function you would write to
       perform the "magic". */
    displayLayer(document.layers[document.numProcessed]);

    /* if there's more work to do, set a timeout to process
       * the next layer. If we're finished, set the document.running
       * flag to false. */
    document.numProcessed = document.numProcessed + 1;
    if (document.numProcessed < document.numLayers){
        setTimeout("processLayer()", 500);
    }else{
        document.running = false;
    }
}
```


A simple floating panel example

The following floating panel contains a text field that shows the contents of the selected Script marker (the yellow icon that appears in the Document window to mark the location of a script). If no Script marker is selected, a layer that contains the text (no script selected) appears.

```
<html>
<head>
<title>Script Editor</title>
<script language="JavaScript">

function selectionChanged(){
    /* get the selected node */
    var theDOM = dw.getDocumentDOM();
    var theNode = theDOM.getSelectedNode();

    /* check to see if the node is a script marker */
    if (theNode.nodeType == Node.ELEMENT_NODE && theNode.tagName == "SCRIPT"){
        document.layers['scriptlayer'].visibility = 'visible';
        document.layers['scriptlayer'].document.theForm.scriptCode.value = theNode.innerHTML;
        document.layers['blanklayer'].visibility = 'hidden';
    }else{
        document.layers['scriptlayer'].visibility = 'hidden';
        document.layers['blanklayer'].visibility = 'visible';
    }
}

/* update the document with any changes made by
   the user in the textarea */
function updateScript(){
    var theDOM = dw.getDocumentDOM();
    var theNode = dw.getSelectedNode();
    theNode.innerHTML = document.layers['scriptlayer'].document.theForm.scriptCode.value;
}

</script>
</head>

<body>
<div id="blanklayer" style="position:absolute; width:422px; height:181px; z-index:1; left: 8px; top: 11px; visibility: hidden">
<center>
<br>
<br>
<br>
<br>
<br>
</div>
```

```

(no script selected)
</center>
</div>

<div id="scriptlayer" style="position:absolute; width:422px;
height:181px; z-index:1; left: 8px; top: 11px; ↵
visibility: visible">
<form name="theForm">
<textarea name="scriptCode" cols="80" rows="20" wrap="VIRTUAL"
onBlur="updateScript()"></textarea>
</form>
</div>

</body>
</html>

```

Remember that it is not sufficient to save this code in a file called `scriptEditor.htm` in the `Configuration/Floater` folder; you must also call `dw.setFloaterVisibility('scriptEditor',true)` or `dw.toggleFloater('scriptEditor')` from somewhere in order to load the floating panel and make it visible. The most obvious place from which to do this is the Window menu in the `menus.xml` file. The `menuitem` tag might look like this:

```

<menuitem name="Script Editor" enabled="true"
command="dw.toggleFloater('scriptEditor')"
checked="dw.getFloaterVisibility('scriptEditor')" />

```

CHAPTER 10

Behaviors

Behaviors let nonprogrammers make their HTML pages interactive. They offer Web designers an easy way to assign actions to page elements by filling in an HTML form.

You should write behavior actions when you want to share functions with nonprogrammers, or when you want to insert the same JavaScript function repeatedly but change the parameters each time.

Note: You cannot use behaviors to insert VBScript functions directly; however, you can add a VBScript function indirectly by editing the DOM in the `applyBehavior()` function.

The term *behavior* refers to the combination of an event (such as `onClick`, `onLoad`, or `onSubmit`) and an action (for example, Check Plugin, Go to URL, Swap Image). The browser determines which HTML elements accept which events. Files listing events that each browser supports are stored in the Configuration/Behaviors/Events folder within the Dreamweaver application folder.

Actions are the part of a behavior you have control over; thus, when you write a behavior, you're really writing an Action file. Actions are HTML files. The BODY of an Action file generally contains an HTML form that accepts parameters for the action (for example, parameters indicating which layers are to be shown or hidden). The HEAD of an Action file contains JavaScript functions that process form input from the BODY and control the functions, arguments, and event handlers that are inserted into a user's document.

How behaviors work

When a user selects an HTML element in a Dreamweaver document and opens the Behaviors panel, the following chain of events occurs:

- 1 The user clicks the Plus (+) button to display the Actions pop-up menu.
- 2 Dreamweaver calls the `canAcceptBehavior()` function in each Action file to see whether this action is appropriate for the document or the selected element. If the return value of this function is `false`, Dreamweaver dims the action in the Actions pop-up menu. (For example, the Control Shockwave action is dimmed when the user's document has no Shockwave movies.) If the return value is a list of events, Dreamweaver compares each event with the valid events for the currently selected HTML element and target browser until it finds a match.
- 3 Dreamweaver populates the Events pop-up menu with the matched event from `canAcceptBehavior()` at the top of the list; if no match exists, the default event for the HTML element (marked in the Event file with an asterisk) becomes the top item. The remaining events in the menu are culled from the Event file.
- 4 The user selects an action from the Actions pop-up menu.
- 5 Dreamweaver calls the `windowDimensions()` function, if defined, to determine the size of the Parameters dialog box. If `windowDimensions()` is not defined, the size is determined automatically.
- 6 Dreamweaver displays a dialog box containing the BODY elements of the Action file. If the Action file's BODY tag contains an `onLoad` handler, Dreamweaver executes it.
- 7 The user fills in the parameters for the action. Dreamweaver executes event handlers associated with the form fields as the user encounters them.
- 8 The user clicks OK.
- 9 Dreamweaver calls the `behaviorFunction()` and `applyBehavior()` functions in the selected Action file. These functions return strings that are inserted into the user's document.
- 10 If the user later double-clicks the action in the Actions column, Dreamweaver reopens the Parameters dialog box, executing the `onLoad` handler. Dreamweaver then calls the `inspectBehavior()` function in the selected Action file, which fills in the fields with the data that the user entered previously.

Inserting multiple functions in the user's file

Actions can insert multiple functions—the main behavior function plus any number of helper functions—into the HEAD. Two or more behaviors can even share helper functions, as long as the function definition is exactly the same in each Action file. One way of ensuring that shared functions are identical is to store each helper function in an external JavaScript file and insert it into the appropriate Action files using `<SCRIPT SRC="externalFile.js">`.

When the user deletes a behavior, Dreamweaver attempts to remove any unused helper functions associated with the behavior. If other behaviors are using a helper function, it will not be deleted. Because the algorithm for deleting helper functions errs on the side of caution, Dreamweaver may occasionally leave behind an unused function in the user's document.

What to do when an action requires a return value

Sometimes an event handler must have a return value (for example, `onMouseOver="window.status='This is a link'; return true'"`). But if Dreamweaver inserts `"return behaviorName(args)"` into the event handler, behaviors later in the list are skipped.

To get around this limitation, set a variable called `document.MM_returnValue` to the desired return value within the string returned by `behaviorFunction()`. This setting causes Dreamweaver to insert `return document.MM_returnValue` at the end of the list of actions in the event handler. See the `Validate Form.js` file in the `Configuration/Behaviors/Actions` folder within the Dreamweaver application folder for an example of the use of `MM_returnValue`.

The behavior API

Two behavior API functions are required (`applyBehavior()` and `behaviorFunction()`); the rest are optional. The functions in the behavior API differ from the functions in the main JavaScript API in three ways:

- They are not methods of the `dreamweaver`, `dom`, or `site` object.
- They are significant only in the context of Behavior files. That is, Dreamweaver automatically calls the `applyBehavior()` function if it is defined in a Behavior file, whereas a function named `applyBehavior()` acts like a user-defined function in any other extension file—you have to call it explicitly.
- You are responsible for writing the body of each function and returning a value, if required. This is the opposite of how the functions work in the main API: those you call and pass arguments to, and Dreamweaver generates return values, if any. Here Dreamweaver calls the functions and passes arguments to them, and you generate return values, if any.

applyBehavior()

Description

Inserts into the user's document an event handler that calls the function inserted by `behaviorFunction()`. This function can also perform other edits on the user's document, but it must not delete the object to which the behavior is being applied or the object that receives the action.

Arguments

uniqueName

The argument is a unique identifier among all instances of all behaviors in the user's document. Its format is *functionNameInteger*, where *functionName* is the name of the function inserted by `behaviorFunction()`. This argument is useful if you insert a tag into the user's document and you want to assign a unique value to its `NAME` attribute.

Returns

A string containing the function call to be inserted in the user's document, usually after accepting parameters from the user. If `applyBehavior()` determines that the user made an invalid entry, the function can return an error string instead of the function call. If the string is empty (`return ""`), Dreamweaver reports no error; but if the string is non-empty and not a function call, Dreamweaver displays a dialog box with the text: Invalid input supplied for this behavior: [the string returned from `applyBehavior()`]. If the return value is null (`return;`), Dreamweaver indicates that an error occurred but offers no specific information.

Note: Quotation marks within the returned string must be preceded by a backslash (\) to avoid errors reported by the JavaScript interpreter.

Example

The following instance of `applyBehavior()` returns a call to the function `MM_openBrWindow()` and passes parameters given by the user (the height and width of the window; whether the window should have scroll bars, a toolbar, a location bar, and other features; and the URL that should open in the window):

```
function applyBehavior() {
    var i,theURL,theName,arrayIndex = 0;
    var argArray = new Array(); //use array to produce correct ↵
    number of commas w/o spaces
    var checkBoxNames = new Array("toolbar","location","status",↵
    "menubar","scrollbars","resizable");

    for (i=0; i<checkBoxNames.length; i++) {
        theCheckBox = eval("document.theForm." + checkBoxNames[i]);
        if (theCheckBox.checked) argArray[arrayIndex++] = ↵
        (checkBoxNames[i] + "=yes");
    }
    if (document.theForm.width.value)
        argArray[arrayIndex++] = ("width=" + ↵
        document.theForm.width.value);
    if (document.theForm.height.value)
        argArray[arrayIndex++] = ("height=" + ↵
        document.theForm.height.value);
    theURL = escape(document.theForm.URL.value);
    theName = document.theForm.winName.value;
    return "MM_openBrWindow('" + theURL + "', '" + theName + "', ↵
    '" + argArray.join(',') + "')";
}
```

behaviorFunction()

Description

Inserts one or more functions—surrounded by <SCRIPT LANGUAGE="JavaScript"></SCRIPT> tags, if none yet exist—into the HEAD of the user's document.

Arguments

None.

Returns

Either a string containing the JavaScript functions to be inserted in the user's document, or a string containing the names of the functions to be inserted into the user's document. This value must be exactly the same every time (it cannot depend on input from the user). The functions are inserted only once, regardless of how many times the action is applied to elements in the document.

Note: Quotation marks within the returned string must be preceded by a backslash (\) to avoid errors reported by the JavaScript interpreter.

Example

The following instance of behaviorFunction() returns a function called MM_popupMsg():

```
function behaviorFunction(){
    return ""+
        "function MM_popupMsg(theMsg) { //v1.0\n"+
        "    alert(theMsg);\n"+
        "    }";
}
```

The following code would be equivalent to the preceding behaviorFunction() declaration, and is the method used to declare behaviorFunction() in all behaviors shipped with Dreamweaver:

```
function MM_popupMsg(theMsg){ //v1.0
    alert(theMsg);
}

function behaviorFunction(){
    return "MM_popupMsg";
}
```


canAcceptBehavior()

Description

Determines whether the action is allowed for the selected HTML element and specifies the default event that should trigger the action. May also check for the existence of certain objects (such as Shockwave movies) in the user's document and disallow the action if these objects do not appear.

Arguments

HTMLElement

The argument is the selected HTML element.

Returns

One of the following values:

- `true` if the action is allowed but has no preferred events.
- A list of preferred events (in descending order of preference) for this action. Specifying preferred events overrides the default event (as denoted with an asterisk in the Event file) for the selected object. See steps 2 and 3 in “How behaviors work” on page 88.
- `false` if the action is not allowed.

If `canAcceptBehavior()` returns `false`, the action is dimmed in the Actions pop-up menu in the Behaviors panel.

Example

The following instance of `canAcceptBehavior()` returns a list of preferred events for the behavior if the document has any named images:

```
function canAcceptBehavior(){
    var theDOM = dreamweaver.getDocumentDOM();
    // Get an array of all images in the document
    var allImages = theDOM.getElementsByTagName('IMG');
    if (allImages.length > 0){
        return "onMouseOver, onClick, onMouseDown";
    }else{
        return false;
    }
}
```

displayHelp()

Description

If this function is defined, a Help button appears below the OK and Cancel buttons in the Parameters dialog box. This function is called when the user clicks the Help button.

Arguments

None.

Returns

Nothing.

Example

The following instance of `displayHelp()` opens, in a browser window, a file with instructions for using the behavior:

```
function displayHelp(){
    dreamweaver.browseDocument('http://www.hotwired.com/~
    webmonkey/javascript/code_library/wm_pos2_elmnt_dw/~
    ?tw=javascript');
}
```

deleteBehavior()

Description

Undoes any edits performed by `applyBehavior()`.

Note: Dreamweaver automatically deletes the function declaration and the event handler associated with a behavior when the user deletes the behavior in the Behaviors panel. Thus, it is necessary to define `deleteBehavior()` only if the `applyBehavior()` function performed additional edits on the user's document (for example, if it inserted a tag).

Arguments

applyBehaviorString

This argument is the string that was returned earlier by the `applyBehavior()` function.

Returns

Nothing.

identifyBehaviorArguments()

Description

Identifies arguments from a behavior function call as `nav`, `dep`, `URL`, `NS4.0ref`, `IE4.0ref`, `objName`, or other so that URLs in behaviors can be updated if the user saves the document to another location, and so that the referenced files can be displayed in the site map and considered dependent files for the purposes of uploading to and downloading from a server.

Arguments

theFunctionCall

This argument is the string that was returned earlier by the `applyBehavior()` function.

Returns

A string containing a comma-separated list of the types of arguments in the function call. The length of the list must equal the number of arguments in the function call. Argument types are always one of the following:

- `nav` specifies that the argument is a navigational URL and therefore that it should be displayed in the site map.
- `dep` specifies that the argument is a dependent file URL and therefore that it should be included with all other dependent files when a document containing this behavior is downloaded from or uploaded to a server.
- `URL` specifies that the argument is both a navigational URL and a dependent URL (or that it is a URL of unknown type), and therefore that it should be displayed in the site map and considered a dependent file when uploading to or downloading from a server.
- `NS4.0ref` specifies that the argument is a Navigator DOM object reference.
- `IE4.0ref` specifies that the argument is an Internet Explorer DOM object reference.
- `objName` specifies that the argument is a simple object name, as specified in the `NAME` attribute for the object. This type was added in Dreamweaver 3.
- `other` specifies that the argument is none of the above types.

Example

This simple example of `identifyBehaviorArguments()` would work for the Open Browser Window behavior action, which returns a function that always has three arguments (the URL to open, the name of the new window, and the list of window properties):

```
function identifyBehaviorArguments(fnCallStr) {  
    return "URL,other,other";  
}
```

A more complex version of `identifyBehaviorArguments()` is necessary for behavior functions that have a variable number of arguments (such as `Show/Hide Layer`). For this version of `identifyBehaviorArguments()`, there is a minimum number of arguments, and additional arguments always come in multiples of the minimum number. In other words, a function with a minimum number of arguments of 4 may have 4, 8, or 12 arguments, but it will never have 10 arguments.

```
function identifyBehaviorArguments(fnCallStr) {
    var listOfArgTypes;
    var itemArray = dreamweaver.getTokens(fnCallStr, '(),');

    // The array of items returned by getTokens() includes the
    // function name, so the number of *arguments* in the array is
    // the length of the array minus one. Divide by 4 to get the
    // number of groups of arguments.
    var numArgGroups = ((itemArray.length - 1)/4);

    //For each group of arguments
    for (i=0; i < numArgGroups; i++){

        // Add a comma and "NS4.0ref,IE4.0ref,other,dep" (because this
        // hypothetical behavior function has a minimum of four
        // arguments: the Netscape object reference, the IE object
        // reference, a dependent URL, and perhaps a property value such
        // as "show" or "hide") to the existing list of argument types,
        // or if no list yet exists, add only
        // "NS4.0ref,IE4.0ref,other,dep"
        var listOfArgTypes += ((listOfArgTypes)?",":"") +
            "NS4.0ref,IE4.0ref,other,dep";
    }
}
```

inspectBehavior()

Description

Inspects the function call for a previously applied behavior in the user's document and sets the values of the options in the Parameters dialog box accordingly. If `inspectBehavior()` is not defined, the default option values appear.

Note: `inspectBehavior()` must rely solely on information passed to it via the *applyBehaviorString* argument. Do not attempt to obtain other information about the user's document (for example, using `dreamweaver.getDocumentDOM()`) within this function.

Arguments

applyBehaviorString

This argument is the string that was returned earlier by the `applyBehavior()` function.

Returns

Nothing.

Example

The following instance of `inspectBehavior()`, taken from `Display Status Message.htm`, fills in the Message field in the parameters form with the message that the user selected when the behavior was originally applied:

```
function inspectBehavior(msgStr){
    var startStr = msgStr.indexOf('"') + 1;
    var endStr = msgStr.lastIndexOf('"');
    if (startStr > 0 && endStr > startStr) {
        document.theForm.message.value =
            unescQuotes(msgStr.substring(startStr,endStr));
    }
}
```

Note: For more information about the `unescQuotes()` function, see the `string.js` file in the `Configuration/Shared/MM/Scripts/CMN` folder.

windowDimensions()

Description

Sets specific dimensions for the Parameters dialog box. If this function is not defined, the window dimensions are computed automatically.

Note: Do not define this function unless you want an Options dialog box larger than 640 x 480 pixels.

Arguments

platform

The value of the argument is either "macintosh" or "windows", depending on the user's platform.

Returns

A string of the form "widthInPixels,heightInPixels".

The returned dimensions are smaller than the size of the entire dialog box because they do not include the area for the OK and Cancel buttons. If the returned dimensions do not accommodate all options, scroll bars appear.

Example

The following instance of `windowDimensions()` sets the dimensions of the Parameters dialog box to 648 x 520 pixels:

```
function windowDimensions(){  
    return "648,520";  
}
```

A simple behavior example

To understand better how behaviors work and how you can create one, it's helpful to look at an example. The Configuration/Behaviors/Actions folder inside the Dreamweaver application folder contains many examples; however, most are likely to be too complex a starting point for all but the most advanced developers. The simplest Action file to start with is `Call JavaScript.htm` (along with its counterpart, `Call JavaScript.js`, which contains all the JavaScript functions).

The following code presents a relatively simple example. It checks the brand of the browser and goes to one page if the brand is Netscape Navigator, and another if the brand is Microsoft Internet Explorer. This code could easily be expanded to check for other brands—such as Opera and WebTV—and modified to perform other actions besides going to URLs.

```

<html>
<head>
<title>behavior "Check Browser Brand"</title>
<meta http-equiv="Content-Type" content="text/html">
<script language="JavaScript">

// The function that will be inserted into the
// HEAD of the user's document
function checkBrowserBrand(netscapeURL,explorerURL) {
    if (navigator.appName == "Netscape") {
        if (netscapeURL) location.href = netscapeURL;
    }else if (navigator.appName == "Microsoft Internet Explorer") {
        if (explorerURL) location.href = explorerURL;
    }
}

//***** API *****/

function canAcceptBehavior(){
    return true;
}
// Return the name of the function to be inserted into
// the HEAD of the user's document
function behaviorFunction(){
    return "checkBrowserBrand";
}

// Create the function call that will be inserted
// with the event handler
function applyBehavior() {
    var nsURL = escape(document.theForm.nsURL.value);
    var ieURL = escape(document.theForm.ieURL.value);
    if (nsURL && ieURL) {
        return "checkBrowserBrand(\"'\" + nsURL + "\"'\",\"'\" + ieURL + "\"'\")";
    }else{
        return "Please enter URLs in both fields."
    }
}

// Extract the arguments from the function call
// in the event handler and repopulate the
// parameters form
function inspectBehavior(fnCall){
    var argArray = getTokens(fnCall, "()",",");
    var nsURL = unescape(argArray[1]);
    var ieURL = unescape(argArray[2]);
    document.theForm.nsURL.value = nsURL;
    document.theForm.ieURL.value = ieURL;
}

```


CHAPTER 11

The Fireworks Integration API

FWLaunch is a C shared library that gives authors of objects, commands, behaviors, and Property inspectors the ability to communicate with Fireworks. This chapter describes the Fireworks integration API and how to use it; for general information on how C libraries interact with the JavaScript interpreter in Dreamweaver, see “C-Level Extensibility” on page 191.

The Fireworks integration API

All functions in the Fireworks integration API are methods of the FWLaunch object. Optional arguments are enclosed in braces ({}).

FWLaunch.bringDWTToFront()

Availability

Dreamweaver 3.0, Fireworks 3.0

Description

Brings Dreamweaver to the front.

Arguments

None.

Returns

Nothing.

FWLaunch.bringFWToFront()

Availability

Dreamweaver 3.0, Fireworks 3.0

Description

Brings Fireworks to the front if it is running.

Arguments

None.

Returns

Nothing.

FWLaunch.execJsInFireworks()

Availability

Dreamweaver 3.0, Fireworks 3.0

Description

Passes the specified string of JavaScript to Fireworks for execution.

Arguments

javascriptOrFileURL

The argument is either a string of literal JavaScript or the path to a .js or .jsf file, expressed as a file:// URL.

Returns

A cookie object if the JavaScript was passed successfully, or a nonzero error code indicating that one of the following errors occurred:

- Invalid usage; *javascriptOrFileURL* was specified as null or empty string, or the path to the .js or .jsf file was invalid.
- File I/O error; Fireworks is unable to create a Response file because the disk is full.
- Error notifying Dreamweaver; the user is not running a valid version of Dreamweaver (3.0 or later).
- Error launching Fireworks process; the function did not launch a valid version of Fireworks (3.0 or later).
- User cancelled the operation.

FWLaunch.getJsResponse()

Availability

Dreamweaver 3.0, Fireworks 3.0

Description

Determines whether Fireworks is still executing the JavaScript passed to it by `FWLaunch.execJsInFireworks()`, whether the script completed successfully, or whether an error occurred.

Arguments

progressTrackerCookie

The argument is the cookie object returned by `FWLaunch.execJsInFireworks()`.

Returns

A string containing the result of the script passed to `FWLaunch.execJsInFireworks()` if the operation completed successfully, `null` if Fireworks is still executing the JavaScript, or a nonzero error code indicating that one of the following errors occurred:

- Invalid usage; a JavaScript error occurred as Fireworks was executing the script.
- File I/O error; Fireworks is unable to create a Response file because the disk is full.
- Error notifying Dreamweaver; the user is not running a valid version of Dreamweaver (3.0 or later).
- Error launching Fireworks process; the function did not launch a valid version of Fireworks (3.0 or later).
- User cancelled the operation.

Returns

The following code passes the string "prompt('Please enter your name:')" to FWLaunch.execJsInFireworks() and then checks for the result:

```
var progressCookie = FWLaunch.execJsInFireworks("prompt('Please enter your name:');");
var doneFlag = false;
while (!doneFlag){
    // check for completion every 1/2 second
    setTimeout('checkForCompletion()',500);
}

function checkForCompletion(){
    if (progressCookie != null) {
        var response = FWLaunch.getJsResponse(progressCookie);
        if (response != null) {
            if (typeof(response) == "number") {
                // error or user-cancel, time to close the window
                // and let the user know we got an error
                window.close();
                alert("An error occurred.");
            }else{
                // got a valid response!
                alert("Nice to meet you, " + response);
                window.close();
            }
            doneFlag = true;
        }
    }
}
```

FWLaunch.mayLaunchFireworks()

Availability

Dreamweaver 2.0, Fireworks 2.0

Description

Determines whether it is possible to launch a Fireworks optimization session.

Arguments

None.

Returns

A Boolean value indicating whether the platform is Windows or Macintosh. If Macintosh, indicates whether another Fireworks optimization session is not already running.

FWLaunch.optimizeInFireworks()

Availability

Dreamweaver 2.0, Fireworks 2.0

Description

Launches a Fireworks optimization session for the specified image.

Arguments

docURL, *imageURL*, {*targetWidth*}, {*targetHeight*}

- The first argument is the path to the active document, expressed as a file:// URL.
- The second argument is the path to the selected image. If the path is relative, it is relative to *docURL*.
- The third argument, if supplied, is the width to which the image should be resized.
- The fourth argument, if supplied, is the height to which the image should be resized.

Returns

0 if a Fireworks optimization session is successfully launched for the specified image; otherwise, a nonzero error code indicating that one of the following errors occurred:

- Invalid usage; *docURL*, *imageURL*, or both were specified as null or empty string.
- File I/O error; Fireworks is unable to create a response file because the disk is full.
- Error notifying Dreamweaver; the user is not running a valid version of Dreamweaver (2.0 or later).
- Error launching Fireworks process; the function did not launch a valid version of Fireworks (2.0 or later).
- User cancelled the operation.

FWLaunch.validateFireworks()

Availability

Dreamweaver 2.0, Fireworks 2.0

Description

Looks for the specified version of Fireworks on the user's hard disk.

Arguments

{versionNumber}

The argument is a floating-point number greater than or equal to 2.0; it represents the version of Fireworks that should be searched for. If this argument is omitted, the default is 2.0.

Returns

A Boolean value indicating whether the specified version of Fireworks was found.

Example

The following code checks whether Fireworks 3.0 is installed:

```
if (FWLaunch.validateFireworks(3.0)){  
    alert( "Fireworks 3.0 is installed.");  
}else{  
    alert( "Fireworks 3.0 is not installed.");  
}
```

A simple Fireworks integration example

The following command asks Fireworks to prompt the user for his or her name, and then returns the name to Dreamweaver.

```
<html>  
<head>  
<title>Prompt in Fireworks</title>  
<meta http-equiv="Content-Type" content="text/html; ↵  
charset=iso-8859-1">  
<script>  
  
function commandButtons(){  
    return new Array("Prompt", "promptInFireworks()", "Cancel", ↵  
        "readyToCancel()", "Close","window.close()");  
}  
  
var gCancelClicked = false;  
var gProgressTrackerCookie = null;  
  
function readyToCancel() {  
    gCancelClicked = true;  
}
```

```

function promptInFireworks() {
    var isFireworks3 = FWLaunch.validateFireworks(3.0);
    if (!isFireworks3) {
        alert("You must have Fireworks 3.0 or later to use this ↵
        command");
    }
    return;
}

// Tell Fireworks to execute the prompt() method.
gProgressTrackerCookie = FWLaunch.execJsInFireworks↵
("prompt('Please enter your name:')");

// null means it wasn't launched, a number means an error code
if (gProgressTrackerCookie == null || ↵
typeof(gProgressTrackerCookie) == "number") {
    window.close();
    alert("an error occurred");
    gProgressTrackerCookie = null;
} else {
    // bring Fireworks to the front
    FWLaunch.bringFWToFront();
    // start the checking to see if Fireworks is done yet
    checkOneMoreTime();
}
}

function checkOneMoreTime() {
    // Call checkJsResponse() every 1/2 second to see if Fireworks
    // is done yet
    window.setTimeout("checkJsResponse();", 500);
}

function checkJsResponse() {
    var response = null;

    // The user clicked the cancel button, close the window
    if (gCancelClicked) {
        window.close();
        alert("cancel clicked");
    } else {
        // We're still going, ask Fireworks how it's doing
        if (gProgressTrackerCookie != null)
            response = ↵
            FWLaunch.getJsResponse(gProgressTrackerCookie);

        if (response == null) {
            // still waiting for a response, call us again in 1/2 a
            // second
            checkOneMoreTime();
        } else if (typeof(response) == "number") {
            // if the response was a number, it means an error
            // occurred

```

```

        // the user cancelled in Fireworks
        window.close();
        alert("an error occurred.");

    } else {
        // got a valid response! This return value might not
        // always be a useful one, since not all functions in
        // Fireworks return a string, but we know this one does,
        // so we can show the user what we got.
        window.close();
        FWLaunch.bringDWToFront();
        // bring Dreamweaver to the front
        alert("Nice to meet you, " + response + "!");
    }
}

</script>
</head>
<body>
<form>
<table width="313" nowrap>
<tr>
<td>This command asks Fireworks to execute the prompt()
function. When you click Prompt, Fireworks comes forward and
asks you to enter a value into a dialog box. That value is then
returned to Dreamweaver and displayed in an alert.</td>
</tr>
</table>
</form>
</body>
</html>

```


CHAPTER 12

The Flash Objects API

The Flash objects API is an extension to the Objects API that allows extension developers to build objects that create simple Flash content. This API provides a way to set parameters in a Flash Generator template (.swt file) and output a Flash Movie or Image file. The API allows you to create new Flash objects as well as read and manipulate existing Flash objects. The Flash button and Flash text features are built using this API.

The .swt file is a Flash Generator Template file, which contains all the necessary information required to construct a Flash Object file (.swf). These API functions allow you to create a new .swf file (or Image file) from a .swt file by replacing the parameters of the .swt file with real values. For more information on Flash and Flash Generator, see their respective manuals.

SWFFile.createFile()

Description

Generates a new Flash Object file with the specified template and array of parameters. Also creates a GIF, PNG, JPEG, and MOV version of the title if file names for those formats are specified.

If you want to specify an optional parameter that comes after optional parameters you do not want to specify, you need to specify empty strings for the parameters you don't want to specify. For example, if you want to specify a .png file, but not a .gif file, you'd need to specify an empty string before specifying the .png file name.

Arguments

templateFile, *templateParams*, *swfFileName*, {*gifFileName*}, {*pngFileName*}, {*jpgFileName*}, {*movFileName*}, {*generatorParams*}

- *templateFile* is a path to a Template file, expressed as a `file:// URL`. This file can be a .swt file.
- *templateParams* is an array of name/value pairs where the names are the names of the parameters in the .swt file and the values are what you want to set those parameters to be. For an .swf file to be recognized by Dreamweaver as a Flash object, the first parameter must be "dwType". Its value should be a string representing the name of the object type, such as "Flash Text".
- *swfFileName* is the output file name of an .swf file name, expressed as a `file:// URL`, or an empty string to ignore.
- {*gifFileName*} is the output file name of a .gif file name, expressed as a `file:// URL`. Optional.
- {*pngFileName*} is the output file name of a .png file name, expressed as a `file:// URL`. Optional.
- {*jpgFileName*} is the output file name of a .jpeg file name, expressed as a `file:// URL`. Optional.
- {*movFileName*} is the output file name of a QuickTime movie file name, expressed as a `file:// URL`. Optional.
- {*generatorParams*} is an array of strings representing optional Generator command line flags. Optional. Each flag must be followed in the array by its data items. Some commonly used flags are listed in the following table.

Option Flag	Data	Description	Example
-defaultsize	width, height	Sets the output image size to the specified width and height.	"-defaultsize", "640", "480"
-exactFit	none	Stretches the contents in the output image to fit exactly into the specified output size.	"-exactFit"

Returns

A string containing one of the following values:

- "noError" means the call completed successfully.
- "invalidTemplateFile" means the specified Template file was invalid or not found.
- "invalidOutputFile" means at least one of the specified output file names was invalid.
- "invalidData" means that one or more of the *templateParams* was invalid.
- "initGeneratorFailed" means Generator could not be initialized.
- "outOfMemory" means insufficient memory to complete the operation.
- "unknownError" means an unknown error occurred.

Example

The following JavaScript creates a Flash Object file of type "myType" that replaces any occurrence of "text" inside the Template file with "Hello World". It will create a .gif file as well as an .swf file.

```
var params = new Array;  
params[0] = "dwType";  
params[1] = "myType";  
params[2] = "text";  
params[3] = "Hello World";  
errorString = SWFFile.createFile( "file:///MyMac/test.swf", "  
params, "file:///MyMac/test.swf", "file:///MyMac/test.gif");
```

SWFFile.getNaturalSize()

Description

Returns the natural size of any Flash movie.

Arguments

fileName

fileName is a path to the Flash movie, expressed as a file:// URL.

Returns

An array containing two elements, representing the width and the height of the movie, or null if the file is not a Flash file.

SWFFile.getObjectType()

Description

Returns the Flash object type—the value that was passed in the `dwType` parameter when the file was created by the `SWFFile.createFile()` function.

Arguments

fileName

fileName is a path to a Flash Object file, expressed as a `file://` URL. This file is usually an `.swf` file.

Returns

A string representing the object type, or `null` if the file is not a Flash Object file or if the file could not be found.

Example

The following code checks to see if the file, `test.swf`, is a Flash object of type `myType`:

```
if ( SWFFile.getObjectType("file:///MyMac/test.swf") == "myType"
){
    alert ("This is a myType object.");
}else{
    alert ("This is not a myType object.");
}
```

SWFFile.readFile()

Description

Reads a Flash Object file.

Arguments

fileName

fileName is a path to a Flash Object file to read, expressed as a `file:// URL`.

Returns

An array of strings. The first array element is the full path to the template .swf file. The following strings represent the parameters (name/value pairs) for the object. Each name is followed in the array by its value. The first name/value pair is "dwType" followed by its value; `null` is returned if the file cannot be found or if it is not a Flash Object file.

Example

Calling this:

```
var params = SWFFile.readFile("file:///MyMac/test.swf");
```

will return the following in the parameters array:

```
"file:///MyMac/test.swf" // the template file used to create this
.swf file
"dwType"                 // first parameter
"myType"                  // first parameter value
"text"                   // second parameter
"Hello World"            // second parameter value
```


CHAPTER 13

The Design Notes API

UltraDev 4, Dreamweaver 4, and Fireworks 4 offer a way for Web designers and developers to store and retrieve extra information about documents—information such as review comments, change notes, or the source file for a GIF or JPEG—in files called Design Notes.

MMNotes is a C shared library that lets extensions authors read and write Design Notes files. Like the DWfile shared library, MMNotes has a JavaScript API that makes it possible to call the functions contained in the library from objects, commands, behaviors, floating panels, Property inspectors, and data translators.

MMNotes also has a C API that gives other applications an opportunity to read and write Design Notes files. The MMNotes shared library can be used independently of Dreamweaver, even if Dreamweaver is not installed.

For more information about using the Design Notes feature from within Dreamweaver, see *Using Dreamweaver*.

How Design Notes work

Each Design Notes file stores information for a single document. If one or more documents in a folder has a Design Notes file associated with it, Dreamweaver creates a `_notes` subfolder where Design Notes files can be stored. The `_notes` folder and the Design Notes files it contains are not visible in the Site window, but they show up in the Finder or Windows Explorer. A Design Notes file name is made up of the main file name plus the extension `.mno`. For example, the Design Notes file associated with `avocado8.gif` is `avocado8.gif.mno`.

Design Notes files are XML files that store information in a series of key/value pairs. The key describes the type of information that is being stored, and the value represents the information itself. Keys are limited to 64 characters.

The Design Notes file for `foghorn.gif` might look like this:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<info>
  <infoitem key="FW_source" value="file:///C:/sites/~
dreamcentral/images/sourceFiles/foghorn.png" />
  <infoitem key="Author" value="Heidi B." />
  <infoitem key="Status" value="Final draft, approved by ~
Jay L." />
</info>
```

The Design Notes JavaScript API

All functions in the Design Notes JavaScript API are methods of the `MMNotes` object. Optional arguments are enclosed in braces (`{}`).

`MMNotes.close()`

Description

Closes the specified Design Notes file and saves any changes. If all key/value pairs were removed, Dreamweaver deletes the Design Notes file.

Arguments

fileHandle

The argument is the file handle returned by `MMNotes.open()`.

Returns

Nothing.

Example

See “`MMNotes.set()`” on page 120.

MMNotes.filePathToLocalURL()

Description

Converts the specified local drive path to a file:// URL.

Arguments

drivePath

The argument is a string containing the full drive path.

Returns

A string containing the file:// URL for the specified file.

Example

A call to `MMNotes.filePathToLocalURL('C:/sites/webdev/index.htm')` returns `"file:///c|sites/webdev/index.htm"`.

MMNotes.get()

Description

Gets the value of the specified key in the specified Design Notes file.

Arguments

fileHandle, keyName

- The first argument is the file handle returned by `MMNotes.open()`.
- The second argument is a string containing the name of the key.

Returns

A string containing the value of the key.

Example

See “`MMNotes.getKeys()`” on page 118.

MMNotes.getKeyCount()

Description

Gets the number of key/value pairs in the specified Design Notes file.

Arguments

fileHandle

The argument is the file handle returned by `MMNotes.open()`.

Returns

An integer representing the number of key/value pairs in the Design Notes file.

MMNotes.getKeys()

Description

Gets a list of all the keys in a Design Notes file.

Arguments

fileHandle

The argument is the file handle returned by `MMNotes.open()`.

Returns

An array of strings, each containing the name of a key.

Example

The following code might be used in a custom floating panel to display the Design Notes information for the active document:

```
var noteHandle = MMNotes.open(dw.getDocumentDOM().URL);
var theKeys = MMNotes.getKeys(noteHandle);
var noteString = "";
var theValue = "";
for (var i=0; i < theKeys.length; i++){
    theValue = MMNotes.get(noteHandle,theKeys[i]);
    noteString += theKeys[i] + " = " theValue + "\n";
}
document.theForm.bigTextField.value = noteString;
```

MMNotes.getSiteRootForFile()

Description

Determines the site root for the specified Design Notes file.

Arguments

fileURL

The argument is the path to a local file, expressed as a file:// URL.

Returns

A string containing the path of the Local Root folder for the site, expressed as a file:// URL, or an empty string if Dreamweaver (or UltraDev) is not installed or the Design Notes file is outside of any site defined with Dreamweaver or UltraDev.

MMNotes.getVersionName()

Description

Gets the version name of the MMNotes shared library, which indicates the application that implemented it.

Arguments

None.

Returns

A string containing the name of the application that implemented the MMNotes shared library.

Example

Calling `MMNotes.getVersionName()` from a Dreamweaver command, object, behavior, Property inspector, floating panel, or data translator returns "Dreamweaver". Calling `MMNotes.getVersionName()` from Fireworks also returns "Dreamweaver" because Fireworks uses the same version of the library (the one created by the Dreamweaver engineering team).

MMNotes.getVersionNum()

Description

Gets the version number of the MMNotes shared library.

Arguments

None.

Returns

A string containing the version number.

MMNotes.localURLToFilePath()

Description

Converts the specified file:// URL to a local drive path.

Arguments

fileURL

The argument is the path to a local file, expressed as a file:// URL.

Returns

A string containing the local drive path for the specified file.

Example

A call to `MMNotes.localURLToFilePath('file:///MacintoshHD/images/moon.gif')` returns "MacintoshHD:images:moon.gif".

MMNotes.open()

Description

Opens the Design Notes file associated with the specified file, or creates one if none exists.

Arguments

filePath, {*bForceCreate*}

- The first argument is the path to the main file with which the Design Notes file is associated, expressed as a file:// URL.
- The second argument is a Boolean value indicating whether to create the Note even if Design Notes is turned off for the site or if *filePath* is not associated with any site.

Returns

The file handle for the Design Notes file, or zero (0) if the file was not opened or created.

Example

See “MMNotes.set()” on page 120.

MMNotes.remove()

Description

Removes the specified key (and its value) from the specified Design Notes file.

Arguments

fileHandle, *keyName*

- The first argument is the file handle returned by `MMNotes.open()`.
- The second argument is a string containing the name of the key to be removed.

Returns

A Boolean value indicating whether the operation was successful.

MMNotes.set()

Description

Creates or updates one key/value pair in a Design Notes file.

Arguments

fileHandle, *keyName*, *valueString*

- The first argument is the file handle returned by `MMNotes.open()`.
- The second argument is a string containing the name of the key.
- The third argument is a string containing the value.

Returns

A Boolean value indicating whether the operation was successful.

Example

The following code opens the Design Notes file associated with a file in the dreamcentral site called peakhike99/index.html, adds a new key/value pair, changes the value of an existing key, and then closes the Note file.

```
var noteHandle = MMNotes.open('file:///c:/sites/dreamcentral/-  
peakhike99/index.html',true);  
MMNotes.set(noteHandle,"Author","M. G. Miller");  
MMNotes.set(noteHandle,"Last Changed","August 28, 1999");  
MMNotes.close(noteHandle);
```

The Design Notes C API

In addition to the JavaScript API, the MMNotes shared library also exposes a C API that lets other applications create Design Notes files. It is not necessary to call these C functions directly if you are using the MMNotes shared library in Dreamweaver; the JavaScript versions of the functions call them for you.

This section contains descriptions of the functions, their arguments, and their return values; you can find definitions for the functions and data types in the MMInfo.h file in the Extending/c_files folder inside the Dreamweaver application folder.

Optional arguments are enclosed in braces ({ }).

void CloseNotesFile()**Description**

Closes the specified Design Notes file and saves any changes. If all key/value pairs were removed from the Note, Dreamweaver deletes it.

Arguments

FileHandle *noteHandle*

The argument is the file handle returned by OpenNotesFile().

Returns

Nothing.

BOOL FilePathToLocalURL()

Description

Converts the specified local drive path to a file:// URL.

Arguments

`const char* drivePath`, `char* localURLBuf`, `int localURLMaxLen`

- The first argument is a string containing the full drive path.
- The second argument is the buffer where the file:// URL should be stored.
- The third argument is the maximum size of `localURLBuf`.

Returns

A Boolean value indicating whether the operation was successful; stores the file:// URL in `localURLBuf`.

BOOL GetNote()

Description

Gets the value of the specified key in the specified Design Notes file.

Arguments

`FileHandle noteHandle`, `const char keyName[64]`, `char* valueBuf`, `int valueBufLength`

- The first argument is the file handle returned by `OpenNotesFile()`.
- The second argument is a string containing the name of the key.
- The third argument is the buffer where the value should be stored.
- The fourth argument is the integer returned by `GetNoteLength(noteHandle, keyName)`, indicating the maximum length of the value buffer.

Returns

A Boolean value indicating whether the operation was successful; stores the value of the key in `valueBuf`.

Example

The following code gets the value of the `comments` key in the Design Notes file associated with `welcome.html`:

```
FileHandle noteHandle = OpenNotesFile("file:///c:/sites/avocado8/~iwjs/welcome.html");
int valueLength = GetNotesLength( noteHandle, "comments");
char* valueBuffer = new char[valueLength + 1]
GetNote(noteHandle, "comments", valueBuffer, valueLength + 1);
printf("Comments: %s",valueBuffer);
CloseNotesFile(noteHandle);
```

int GetNoteLength()

Description

Gets the length of the value associated with the specified key.

Arguments

FileHandle *noteHandle*, const char *keyName*[64]

- The first argument is the file handle returned by `OpenNotesFile()`.
- The second argument is a string containing the name of the key.

Returns

An integer representing the length of the value.

Example

See “`BOOL GetNote()`” on page 122.

int GetNotesKeyCount()

Description

Gets the number of key/value pairs in the specified Design Notes file.

Arguments

FileHandle *noteHandle*

The argument is the file handle returned by `OpenNotesFile()`.

Returns

An integer representing the number of key/value pairs in the Design Notes file.

BOOL GetNotesKeys()

Description

Gets a list of all the keys in a Design Notes file.

Arguments

FileHandle *noteHandle*, char* *keyBufArray*[64], int *keyArrayMaxLen*

- The first argument is the file handle returned by `OpenNotesFile()`.
- The second argument is the buffer array where the keys should be stored.
- The third argument is the integer returned by `GetNotesKeyCount(noteHandle)`, indicating the maximum number of items in the key buffer array.

Returns

A Boolean value indicating whether the operation was successful; stores the key names in *keyBufArray*.

Example

The following code prints the key names and values of all the keys in the Design Notes file associated with `welcome.html`:

```
typedef char[64] InfoKey;
FileHandle noteHandle = OpenNotesFile("file:///c:/sites/avocado8/~
iwjs/welcome.html");
if (noteHandle > 0){
    int keyCount = GetNotesKeyCount(noteHandle);
    if (keyCount <= 0)
        return;
    InfoKey* keys = new InfoKey[keyCount];
    BOOL succeeded = GetNotesKeys(noteHandle, keys, keyCount);

    if (succeeded){
        for (int i=0; i < keyCount; i++){
            printf("Key is: %s\n", keys[i]);
            printf("Value is: %s\n\n", GetNote(noteHandle, keys[i]));
        }
        delete keys;
    }
    CloseNotesFile(noteHandle);
}
```


BOOL GetSiteRootForFile()

Description

Determines the site root for the specified Design Notes file.

Arguments

char filePath*, *char* siteRootBuf*, *int siteRootBufMaxLen*, {*InfoPrefs* infoPrefs*}

- The first argument is the file handle returned by `OpenNotesFile()`.
- The second argument is the buffer where the site root should be stored.
- The third argument is the maximum size of *siteRootBuf*.
- The fourth argument is a reference to a struct in which the preferences for the site should be stored.

Returns

A Boolean value indicating whether the operation was successful; stores the site root in *siteRootBuf*. If *infoPrefs* is specified, the function also returns the Design Notes preferences for the site. The `InfoPrefs` struct has two variables: `bUseDesignNotes` and `bUploadDesignNotes`, both of type `BOOL`.

BOOL GetVersionName()

Description

Gets the version name of the MMNotes shared library, which indicates the application that implemented it.

Arguments

char versionNameBuf*, *int versionNameBufMaxLen*

- The first argument is the buffer where the version name should be stored.
- The second argument is the maximum size of *versionNameBuf*.

Returns

A Boolean value indicating whether the operation was successful; stores the version name in *versionNameBuf*.

BOOL GetVersionNum()

Description

Gets the version number of the MMNotes shared library.

Arguments

`char* versionNumBuf, int versionNumBufMaxLen`

- The first argument is the buffer where the version number should be stored.
- The second argument is the maximum size of *versionNumBuf*.

Returns

A Boolean value indicating whether the operation was successful; stores the version number in *versionNumBuf*.

BOOL LocalURLToFilePath()

Description

Converts the specified file:// URL to a local drive path.

Arguments

`const char* localURL, char* drivePathBuf, int drivePathMaxLen`

- The first argument is the path to a local file, expressed as a file:// URL.
- The second argument is the buffer where the local drive path should be stored.
- The third argument is the maximum size of *drivePathBuf*.

Returns

A Boolean value indicating whether the operation was successful; stores the local drive path in *drivePathBuf*.

FileHandle OpenNotesFile()

Description

Opens the Design Notes file associated with the specified file, or creates one if none exists.

Arguments

`const char* localFileURL, {BOOL bForceCreate}`

- The first argument is a string containing the path to the main file with which the Design Notes file is associated, expressed as a file:// URL.
- The second argument is a Boolean value indicating whether to create the Design Notes file even if Design Notes is turned off for the site or if *filePath* is not associated with any site.

FileHandle OpenNotesFilewithOpenFlags()

Description

Opens the Design Notes file associated with the specified file, or creates one if none yet exists. You can open the file in read-only mode.

Arguments

`const char* localFileURL, {BOOL bForceCreate}, {BOOL bReadOnly}`

- The first argument is a string containing the path to the main file with which the Design Notes file is associated, expressed as a file:// URL.
- The second argument is a Boolean value indicating whether to create the Design Notes file even if Design Notes are turned off for the site or *filePath* is not associated with any site. The default value is `false`. This argument is optional, though you need to specify it if you specify the third argument.
- The third argument is a Boolean value indicating whether to open the file in read-only mode. The default value is `false`. This argument is optional.

BOOL RemoveNote()

Description

Removes the specified key (and its value) from the specified Design Notes file.

Arguments

`FileHandle noteHandle, const char keyName[64]`

- The first argument is the file handle returned by `OpenNotesFile()`.
- The second argument is a string containing the name of the key to be removed.

Returns

A Boolean value indicating whether the operation was successful.

BOOL SetNote()

Description

Creates or updates one key/value pair in a Design Notes file.

Arguments

`FileHandle noteHandle, const char keyName[64], const char* value`

- The first argument is the file handle returned by `OpenNotesFile()`.
- The second argument is a string containing the name of the key.
- The third argument is a string containing the value.

Returns

A Boolean value indicating whether the operation was successful.

CHAPTER 14

The File I/O API

Dreamweaver includes a C shared library called DWfile that gives authors of objects, commands, behaviors, data translators, floating panels, and Property inspectors the ability to read and write files on the local file system. This chapter describes the file I/O API and how to use it.

For general information on how C libraries interact with the JavaScript interpreter in Dreamweaver, see “C-Level Extensibility” on page 191.

Verifying that DWfile is installed

To access the functions in the DWfile library, the library must be present in the Configuration/JSExtensions folder and loaded by Dreamweaver. Because DWfile did not ship with Dreamweaver 2 (it was available as a separate download that the user had to install), you should verify that the library is available before calling any of its functions. You can do this by checking the `typeof(DWfile)`. If DWfile does not exist, `typeof(DWfile)` returns `undefined`. For example, if you are using DWfile in the context of a command, you might check for the existence of DWfile as part of the `canAcceptCommand()` function:

```
// Returns true if typeof(DWfile) is not undefined, false otherwise.
function canAcceptCommand(){
    return (typeof(DWfile) != "undefined");
}
```

The file I/O API

All functions in the file I/O API are methods of the `DWfile` object. Optional arguments are enclosed in braces (`{ }`). Functions with an availability of 2.0 were included in the version of `DWfile` that was supplied as a download for Dreamweaver 2 from the Macromedia Web site. This version of `DWfile` may have been installed with third-party objects.

`DWfile.copy()`

Availability

Dreamweaver 3.0

Description

Copies the specified file to the specified URL.

Arguments

originalURL, *copyURL*

- The first argument is the file you want to copy, expressed as a `file://` URL.
- The second argument is the location where you save the copied file, expressed as a `file://` URL.

Returns

`true` if the copy succeeded, `false` otherwise.

Example

The following code copies a file called `myconfig.cfg` to `myconfig_backup.cfg`:

```
var fileURL = "file:///c:/Config/myconfig.cfg";  
var newURL = "file:///c:/Config/myconfig_backup.cfg";  
DWfile.copy(fileURL, newURL);
```

`DWfile.createFolder()`

Availability

Dreamweaver 2.0

Description

Creates a folder (directory) at the specified location.

Arguments

folderURL

The argument is the location of the folder you want to create, expressed as a `file://` URL.

Returns

`true` if the folder was successfully created, `false` otherwise.

Example

The following code attempts to create a folder called tempFolder at the top level of the C drive and displays an alert box indicating whether the operation was successful.

```
var folderURL = "file:///c:/tempFolder";
if (DWfile.createFolder(folderURL)){
    alert("Created " + folderURL);
}else{
    alert("Unable to create " + folderURL);
}
```

DWfile.exists()

Availability

Dreamweaver 2.0

Description

Tests for the existence of the specified file.

Arguments

fileURL

The argument is the file you are looking for, expressed as a file:// URL.

Returns

true if the file exists, false otherwise.

Example

The following code checks for a file called mydata.txt and displays an alert box that tells the user whether the file exists.

```
var fileURL = "file:///c:/temp/mydata.txt";
if (DWfile.exists(fileURL)){
    alert( fileURL + " exists!");
}else{
    alert( fileURL + " does not exist.");
}
```

DWfile.getAttributes()

Availability

Dreamweaver 2.0

Description

Gets the attributes of the specified file or folder.

Arguments

fileURL

The argument is the file or folder for which you want to get attributes, expressed as a file:// URL.

Returns

A string representing the attributes of the specified file or folder, or `null` if the file or folder does not exist. Characters in the string represent the attributes as follows:

- R is read only.
- D is folder (directory).
- H is hidden.
- S is system file or folder.

Example

The following code gets the attributes of the file `mydata.txt` and displays an alert box if the file is read only:

```
var URL = "file:///c:/temp/mydata.txt";
var str = DWfile.getAttributes(URL);
if (str && (str.indexOf("R") != -1)){
    alert(URL + " is read only!");
}
```


DWfile.getModificationDate()

Availability

Dreamweaver 2.0

Description

Gets the time when the file was last modified.

Arguments

fileURL

The argument is the file for which you are checking the last modified time, expressed as a file:// URL.

Returns

A string containing a hexadecimal number that represents the number of time units that have elapsed since some base time. The exact meaning of time units and base time is platform-dependent; in Windows, for example, a time unit is 100ns, and the base time is January 1st, 1600.

Example

It's most useful to call the function twice and compare the return values because the value returned by this function is platform-dependent and is not a recognizable date and time. For example, the following code gets the modification dates of file1.txt and file2.txt and displays an alert box indicating which file is newer:

```
var file1 = "file:///c:/temp/file1.txt";
var file2 = "file:///c:/temp/file2.txt";
var time1 = DWfile.getModificationDate(file1);
var time2 = DWfile.getModificationDate(file2);
if (time1 == time2){
    alert("file1 and file2 were saved at the same time");
}else if (time1 < time2){
    alert("file1 older than file2");
}else{
    alert("file1 is newer than file2");
}
```

DWfile.getCreationDate()

Availability

Dreamweaver 4.0

Description

Gets the time when the file was created.

Arguments

fileURL

The argument is the file for which you are checking the creation time, expressed as a file:// URL.

Returns

A string containing a hexadecimal number that represents the number of time units that have elapsed since some base time. The exact meaning of time units and base time is platform-dependent; in Windows, for example, a time unit is 100ns, and the base time is January 1st, 1600.

Example

You can call this function and the DWfile.getModificationDate() function on a file to compare the modification date to the creation date:

```
var file1 = "file:///c:/temp/file1.txt";
var time1 = DWfile.getCreationDate(file1);
var time2 = DWfile.getModificationDate(file1);
if (time1 == time2){
    alert("file1 has not been modified since it was created");
}else if (time1 < time2){
    alert("file1 was last modified on time2");
}
```

DWfile.listFolder()

Availability

Dreamweaver 2.0

Description

Gets a list of the contents of the specified folder.

Arguments

folderURL, {*constraint*}

- The first argument is the folder for which you want a contents list, expressed as a file:// URL, plus an optional wildcard file mask. Valid wildcards are * (matches 1 or more characters) and ? (matches a single character).
- The second argument, if supplied, must be either "files" (return only files) or "directories" (return only directories). If omitted, the function returns both files and directories.

Returns

An array of strings representing the contents of the folder.

Example

The following code gets a list of all the text (.txt) files in the temp folder and displays the list in an alert box:

```
var folderURL = "file:///c:/temp";
var fileMask = "*.txt";
var list = listFolder(folderURL + "/" + fileMask, "files");
if (list){
    alert(folderURL + " contains: " + list.join("\n"));
}
```

DWfile.read()**Availability**

Dreamweaver 2.0

Description

Reads the contents of the specified file into a string.

Arguments

fileURL

The argument is the file you want to read, expressed as a file:// URL.

Returns

A string containing the contents of the file, or null if the read fails.

Example

The following code reads the file mydata.txt and, if successful, displays an alert box with the contents of the file:

```
var fileURL = "file:///c:/temp/mydata.txt";
var str = DWfile.read( fileURL);
if (str){
    alert( fileURL + " contains: " + str);
}
```

DWfile.remove()

Availability

Dreamweaver 3.0

Description

Moves the specified file to the Recycling Bin or Trash.

Arguments

fileURL

The argument is the file you want to remove, expressed as a file:// URL.

Returns

true if the operation succeeded, false otherwise.

DWfile.write()

Availability

Dreamweaver 2.0

Description

Writes the specified string to the specified file. If the specified file does not yet exist, it is created.

Arguments

fileURL, text, {mode}

- The first argument is the file you are writing to, expressed as a file:// URL.
- The second argument is the string to be written.
- The third argument, if supplied, must be "append". If this argument is omitted, the contents of the file are overwritten by the string.

Returns

true if the string was successfully written to the file, false otherwise.

Example

The following code attempts to write the string "xxx" to the file mydata.txt and displays an alert if the write succeeded. It then attempts to append the string "aaa" to the file and displays a second alert if the write succeeded. After executing this script, the file mydata.txt will contain the text xxxaaa and nothing else.

```
var fileURL = "file:///c:/temp/mydata.txt";
if (DWfile.write(fileURL, "xxx")){
    alert("Wrote xxx to " + fileURL);
}
if (DWfile.write(fileURL, "aaa", "append")){
    alert("Appended aaa to " + fileURL);
}
```

CHAPTER 15

The HTTP API

Extensions are not limited to working within the local file system. Dreamweaver provides a mechanism for getting information from and sending information to a Web server via hypertext transfer protocol (HTTP). This chapter describes the HTTP API and how to use it.

The HTTP API

All functions in the HTTP API are methods of the `MMHttp` object. Most take at least a URL as an argument, and most return an object. The default port for URL arguments is 80; to specify a port other than 80, append a colon and the port number to the URL. For example:

```
MMHttp.getText("http://www.myserver.com:8025");
```

For functions that return an object, the object has two properties: `statusCode` and `data`.

`statusCode` indicates the status of the operation; possible values include but are not limited to:

- 200: Status OK
- 400: Unintelligible request
- 404: Requested URL not found
- 405: Server does not support requested method
- 500: Unknown server error
- 503: Server capacity reached

For a comprehensive list of status codes for your server, check with your Internet service provider or system administrator.

The value of the `data` property varies according to the function; possible values are specified in the individual function listings.

Functions that return an object also have a “callback” version. Callback functions allow other functions to execute while the Web server processes an HTTP request. This is useful if you are making multiple HTTP requests from Dreamweaver. The callback version of a function passes its ID and return value directly to the function specified as its first argument.

Optional arguments are enclosed in braces (`{ }`).

MMHttp.clearTemp()

Description

Deletes all files in the `Configuration/Temp` folder inside the Dreamweaver application folder.

Arguments

None.

Returns

Nothing.

Example

The following code, when saved in a file inside the `Configuration/Shutdown` folder, removes all files from the `Configuration/Temp` folder when the user quits Dreamweaver:

```
<html>
<head>
<title>Clean Up Temp Files on Shutdown</title>
</head>
<body onload="MMHttp.clearTemp()">
</body>
</html>
```

MMHttp.getFile()

Description

Gets the file at the specified URL and saves it in the `Configuration/Temp` folder inside the Dreamweaver application folder on the user's hard disk. Dreamweaver automatically creates subfolders that mimic the folder structure of the server; for example, if the specified file is at `http://www.dreamcentral.com/people/index.html`, Dreamweaver stores the `index.html` file in the `People` folder inside the `www.dreamcentral.com` folder.

Arguments

URL, {prompt}, {saveURL}, {titleBarLabel}

- The first argument is an absolute URL on a Web server; if “http://” is omitted from the URL, it is assumed.
- The second argument is a Boolean value that specifies whether to prompt the user to save the file. If *saveURL* is outside the Configuration/Temp folder, a *prompt* value of *false* is ignored for security reasons.
- The third argument is the location on the user’s hard disk where the file should be saved, expressed as a file:// URL. If *prompt* is *true* or *saveURL* is outside the Configuration/Temp folder, the user can override *saveURL* in the Save dialog box.
- The fourth argument is the label that should appear in the title bar of the Save dialog box.

Returns

An object that represents the reply from the server. The *data* property of this object is a string containing the location where the file was saved, expressed as a file:// URL. Normally the *statusCode* property of the object contains the status code received from the server. However, if a disk error occurs while Dreamweaver is saving the file on the local drive, the *statusCode* property contains an integer representing one of the following error codes if the operation was not successful:

- 1: Unspecified error
- 2: File not found
- 3: Invalid path
- 4: Number of open files limit reached
- 5: Access denied
- 6: Invalid file handle
- 7: Cannot remove current working directory
- 8: No more directory entries
- 9: Error setting file pointer
- 10: Hardware error
- 11: Sharing violation
- 12: Lock violation
- 13: Disk full
- 14: End of file reached

Example

The following code gets an HTML file, saves all the files in the Configuration/Temp folder, and then opens the local copy of the HTML file in a browser:

```
var httpReply = MMHttp.getFile("http://www.dreamcentral.com/~people/profiles/scott.html", false);
if (httpReply.statusCode == 200){
    var saveLoc = httpReply.data;
    dw.browseDocument(saveLoc);
}
```

MMHttp.getFileCallback()

Description

Gets the file at the specified URL, saves it in the Configuration/Temp folder inside the Dreamweaver application folder on the user's hard disk, and then calls the specified function with the request ID and reply result. When saving the file locally, Dreamweaver automatically creates subfolders that mimic the directory structure of the server; for example, if the specified file is at `http://www.dreamcentral.com/people/index.html`, Dreamweaver stores the `index.html` file in the People folder inside the `www.dreamcentral.com` folder.

Arguments

callbackFunction, URL, {prompt}, {saveURL}, {titleBarLabel}

- The first argument is the name of the JavaScript function to call when the HTTP request is complete.
- The second argument is an absolute URL on a Web server; if “http://” is omitted from the URL, it is assumed.
- The third argument is a Boolean value that specifies whether to prompt the user to save the file. If *saveURL* is outside the Configuration/Temp folder, a *prompt* value of *false* is ignored for security reasons.
- The fourth argument is the location on the user's hard disk where the file should be saved, expressed as a file:// URL. If *prompt* is *true* or *saveURL* is outside the Configuration/Temp folder, the user can override *saveURL* in the Save dialog box.
- The fifth argument is the label that should appear in the title bar of the Save dialog box.

Returns

An object that represents the reply from the server. The *data* property of this object is a string containing the location where the file was saved, expressed as a file:// URL. Normally the *statusCode* property of the object contains the status code received from the server. However, if a disk error occurs while Dreamweaver is saving the file on the local drive, the *statusCode* property contains an integer representing an error code. See “MMHttp.getFile()” on page 138 for a list of possible error codes.

MMHttp.getText()

Description

Retrieves the contents of the document at the specified URL.

Arguments

URL

The argument is an absolute URL on a Web server; if “http://” is omitted from the URL, it is assumed.

Returns

An object that represents the reply from the server. The `data` property of this object is a string containing the contents of the document.

Example

The following code gets the contents of a file on a Web server and puts it in a new, untitled Dreamweaver document:

```
var httpReply = MMHttp.getText("http://www.dreamcentral.com/~people/profiles/lori.html");
if (httpReply.statusCode == 200){
    var newDoc = dw.createDocument();
    newDoc.documentElement.outerHTML = httpReply.data;
}
```

MMHttp.getTextCallback()

Description

Retrieves the contents of the document at the specified URL and passes it to the specified function.

Arguments

callbackFunc, *URL*

- The first argument is the name of the JavaScript function to call when the HTTP request is complete.
- The second argument is an absolute URL on a Web server; if “http://” is omitted from the URL, it is assumed.

Returns

An object that represents the reply from the server. The data property of this object is a string containing the contents of the document.

Example

The following code populates a form field with the text returned by the MMHttp.GetTextCallback() function or shows an error message if the function returns an error:

```
var requestID = MMHttp.getTextCallback("httpCallback", "www.dreamcentral.com/index.html")

function httpCallback(requestID,reply) {
    if (reply.statusCode == 200) {
        document.theForm.docContents.value = reply.data;
    }else{
        alert("Request #: " + requestID + "returned the following error: " + reply.statusCode);
    }
}
```

MMHttp.postText()

Description

Performs an HTTP post of the specified data to the specified URL. Typically the data associated with a post operation is form-encoded text, but it could be any type of data that the server expects to receive.

Arguments

URL, dataToPost, {contentType}

- The first argument is an absolute URL on a Web server; if "http://" is omitted from the URL, it is assumed.
- The second argument is the data to be posted. If the third argument is "application/x-www-form-urlencoded" or omitted, *dataToPost* must be form-encoded according to section 8.2.1 of the RFC 1866 specification (available at <http://www.faqs.org/rfcs/rfc1866.html>).
- The third argument is the content type of the data to be posted. If omitted, this argument defaults to "application/x-www-form-urlencoded".

Returns

An object that represents the reply from the server. The *data* property of this object is a string containing the data resulting from the post operation.

MMHttp.postTextCallback()

Description

Performs an HTTP post of the text to the specified URL and passes the reply from the server to the specified function. Typically the data associated with a post operation is form-encoded text, but it could be any type of data that the server expects to receive.

Arguments

callbackFunc, *URL*, *dataToPost*, [*contentType*]

- The first argument is the name of the JavaScript function to call when the HTTP request is complete.
- The second argument is an absolute URL on a Web server; if “http://” is omitted from the URL, it is assumed.
- The third argument is the data to be posted. If the third argument is “application/x-www-form-urlencoded” or omitted, *data* must be form-encoded according to section 8.2.1 of the RFC 1866 specification (available at <http://www.faqs.org/rfcs/rfc1866.html>).
- The fourth argument is the content type of the data to be posted. If omitted, this argument defaults to “application/x-www-form-urlencoded”.

Returns

An object that represents the reply from the server. The *data* property of this object is a string containing the data resulting from the post operation.

CHAPTER 16

The Database API

Database functions allow you to work with structured query language (SQL) statements and stored procedures. Using these functions, you can retrieve certain database schema information. A database schema is the structure of a database. (This structural information is also referred to as metadata.) The structure of a database includes the database's table and column names. With the database API functions, you can get table and column names from SQL statements and stored procedures, get user names and passwords used to establish database connections, and show the results of an executed SQL statement or stored procedure. These functions are used at design time when users are building their Web applications, as opposed to run time, when the Web application is deployed.

The database functions can be used by any extension. In fact, the UltraDev server behavior, data format, and data source API functions make use of the database functions.

The following example shows how the server behavior function, `getDynamicBindings()`, is defined for Recordset. (The file, `Recordset.js`, is located in the `/Configuration/ServerBehaviors/ASP` folder.)

Notice that the `MMDB.getColumnList()` function is used.

```
function getDynamicBindings(elementNode)
{
    var ss = findSSrec(elementNode, LABEL_Type)

    var connString = ss.activeconnection
    var connName = ss.connectionName
    var statement = ss.source
    var rsName = ss.rsName

    var pa = new Array()

    if (String(ss.ParamArray) != "undefined")
    {
        for (var i = 0; i < ss.ParamArray.length; i++)
        {
            pa[i] = new Array()
            pa[i][0] = ss.ParamArray[i].name
            pa[i][1] = ss.ParamArray[i].value
        }
    }

    var statement = ReplaceParamsWithVals(statement, pa)
    return MMDB.getColumnList(connName, statement)
}
```

Database API functions

The following list describes some of the arguments common to the database functions:

- Most database functions use a connection name as an argument. You can see a list of valid connection names in the UltraDev Connection Manager, or you can use `MMDB.getConnectionList()` to get a list of all the connection names programmatically.
- Stored procedures often require parameters. There are two ways of specifying parameter values for database functions. First, you can provide an array of parameter values (*paramValuesArray*). If you specify only parameter values, the values need to be in the sequence in which the stored procedure requires the parameters. Second, you specify parameter values to provide an array of parameter names (*paramNameArray*). (You can use the `MMDB.getSPParamsAsString()` function to get the parameters of the stored procedure.) If you provide parameter names, the values specified in *paramValuesArray* need to be in the sequence in which the parameter names were specified in *paramNameArray*.

getColdFusionDsnList()

Availability

Dreamweaver UltraDev 4.0

Description

Gets the ColdFusion DSNs from the site server, using the `getRdsUserName()` and `getRdsPassword()` functions.

Arguments

None.

Returns

An array containing the ColdFusion DSNs that are defined on the server for the current site.

MMDB.getColumnAndTypeList()

Availability

Dreamweaver UltraDev 1.0

Description

Gets a list of columns and their types from an executed SQL SELECT statement.

Arguments

connName, *statement*

- *connName* is the UltraDev connection name as specified in the Connection Manager. It is used to make a database connection to a live data source.
- *statement* is the SQL SELECT statement to execute.

Returns

An array of strings representing a list of columns (and their types) that match the SELECT statement, or an error if the SQL statement was invalid or the connection could not be made.

Example

The code `var columnArray =`

```
MMDB.getColumnAndTypeList("EmpDB","Select * from Employees")
```

returns the following array of strings:

```
columnArray[0] = "EmpName" , columnArray[1] = "varchar", ↵  
columnArray[2] = "EmpFirstName", columnArray[3] = "varchar", ↵  
columnArray[4] = "Age", columnArray[5] = "integer"
```

MMDB.getColumnList()

Availability

Dreamweaver UltraDev 1.0

Description

Gets a list of columns from an executed SQL SELECT statement.

Arguments

connName, statement

- *connName* is an UltraDev connection name as specified in the Connection Manager. It is used to make a database connection to a live data source.
- *statement* is the SQL SELECT statement to execute.

Returns

An array of strings representing a list of columns that match the SELECT statement, or an error if the SQL statement was invalid or the connection could not be made.

Example

The code `var columnArray = MMDB.getColumnList("EmpDB","Select * from Employees")` returns the following array of strings:

```
columnArray[0] = "EmpName", columnArray[1] = "EmpFirstName", ↵  
columnArray[2] = "Age"
```


MMDB.getColumnsOfTable()

Availability

Dreamweaver UltraDev 1.0

Description

Gets a list of all the columns in the specified table.

Arguments

connName, *tableName*

- *connName* is the UltraDev connection name as specified in the Connection Manager. It is used to make a database connection to a live data source.
- *tableName* is the name of a table in the database specified by *connName*.

Returns

An array of strings; each string is the name of a column in the table.

Example

The statement `MMDB.getColumnsOfTable ("EmpDB","Employees");` returns the following strings:

```
["EmpID", "FirstName", "LastName"]
```

MMDB.getConnectionList()

Availability

Dreamweaver UltraDev 1.0

Description

Gets a list of all the connection strings defined in the Connection Manager.

Arguments

None.

Returns

An array of strings; each string is the name of a connection as it appears in the Connection Manager.

Example

A call to `MMDB.getConnectionList()` could return the strings `["EmpDB", "Test", "TestEmp"]`.

MMDB.getConnectionName()

Availability

Dreamweaver UltraDev 1.0

Description

Gets the connection name corresponding to the specified connection string. This function is useful when you need to reselect a connection name in the user interface from data on the page.

If you have a connection string that references two different drivers, you can specify both the connection string and the driver that corresponds to the connection name you want returned. For example, you could have two connections:

Connection1 has the following properties:

```
ConnectionString="jdbc:inetdae:velcro-qa-5:1433?database=pubs"  
DriverName="com.inet.tds.TdsDriver"
```

Connection2 has the following properties:

```
ConnectionString="jdbc:inetdae:velcro-qa-5:1433?database=pubs"  
DriverName="com.inet.tds.TdsDriver2"
```

The connection strings for both Connection1 and Connection2 are the same. Connection2 connects to a more recent version of TdsDriver. You should pass the driver name to this function to fully qualify the connection name you would like returned.

Arguments

connString, {*driverName*}

- *connString* is the connection string used to get the connection name.
- *driverName* is an optional argument that further qualifies *connString*.

Returns

A connection name string corresponding to the connection string.

Example

The following code returns the string "EmpDB":

```
var connectionName = MMDB.getConnectionName (  
  ("dsn=EmpDB;uid=;pwd=");
```

MMDB.getConnectionString()

Availability

Dreamweaver UltraDev 1.0

Description

Gets the connection string associated with the specified connection name.

Arguments

connName

connName is an UltraDev connection name as specified in the Connection Manager. It is used to make a database connection to a live data source.

Returns

A string corresponding to the connection name.

Example

The code `var connectionString = MMDB.getConnectionString ("EmpDB")` returns different strings for an ADO or JDBC connection.

For an ADO connection, the following string could be returned:

```
"dsn=EmpDB;uid=;pwd=";
```

For a JDBC connection, the following string could be returned:

```
"jdbc:inetdae:192.168.64.49:1433?database=pubs&user=JoeUser&-password=joesSecret"
```

MMDB.getDriverName()

Availability

Dreamweaver UltraDev 1.0

Description

Gets the driver name associated with the specified connection. Only a JDBC connection has a driver name in Dreamweaver UltraDev 1.0.

Arguments

connName

connName is an UltraDev connection name as specified in the Connection Manager. It is used to make a database connection to a live data source.

Returns

A string containing the driver name.

Example

The statement `MMDB.getDriverName ("EmpDB")`; might return the following string:

```
"jdbc/oracle/driver/JdbcOracle"
```

MMDB.getPassword()

Availability

Dreamweaver UltraDev 1.0

Description

Gets the password used for the specified connection.

Arguments

connName

connName is an UltraDev connection name as specified in the Connection Manager. It is used to make a database connection to a live data source.

Returns

A password string associated with the connection name.

Example

The statement `MMDB.getPassword ("EmpDB");` might return "joessecret".

MMDB.getRuntimeConnectionType()

Availability

Dreamweaver UltraDev 1.0

Description

Returns the run-time connection type of the specified connection name. This function can return one of the following values: "ADO", "ADODSN", "JDBC", or "CFDSN".

Arguments

connName

connName is the UltraDev connection name as specified in the Connection Manager. It is used to make a database connection to a live data source.

Returns

A string corresponding to the connection type.

Example

The following code would return the string "ADO" for an ADO connection:

```
var connectionType = MMDB.getRuntimeConnectionType ("EmpDB")
```

MMDB.getSPColumnList()

Availability

Dreamweaver UltraDev 1.0

Description

Gets a list of result set columns that are generated by a call to the specified stored procedure.

Arguments

connName, *statement*, *paramValuesArray*

- *connName* is an UltraDev connection name as specified in the Connection Manager. It is used to make a database connection to a live data source.
- *statement* is the name of the stored procedure that returns the result set when executed.
- *paramValuesArray* is an array containing a list of design-time parameter test values. Specify the parameter values in the order in which the stored procedure expects them. You can use the MMDB.getSPParamsAsString() function to get the parameters of the stored procedure.

Returns

An array of strings representing the list of columns. This function returns an error if the SQL statement is invalid or if the connection string is wrong.

Example

The following code could return a list of result set columns that were generated from the executed stored procedure, getNewEmployeesMakingAtLeast:

```
var paramValueArray = new Array("2/1/2000", "50000")
var columnArray = MMDB.getSPColumnList("EmpDB", ↵
"getNewEmployeesMakingAtLeast", paramValueArray)
```

These are the returned values:

```
columnArray[0] = "EmpID", columnArray[1] = "LastName", ↵
columnArray[2] = "startDate", columnArray[3] = "salary"
```

MMDB.getSPColumnListNamedParams()

Availability

Dreamweaver UltraDev 1.0

Description

Gets a list of result set columns that are generated by a call to the specified stored procedure.

Arguments

connName, *statement*, *paramNameArray*, *paramValuesArray*

- *connName* is an UltraDev connection name as specified in the Connection Manager. It is used to make a database connection to a live data source.
- *statement* is the name of the stored procedure that returns the result set when executed.
- *paramNameArray* is an array containing a list of parameter names. You can use the `MMDB.getSPParamsAsString()` function to get the parameters of the stored procedure.
- *paramValuesArray* is an array containing a list of design-time parameter test values. Optional, specify if the procedure requires parameters when executed. If you have provided parameter names in *paramNameArray*, specify the parameter values in the same order as their corresponding parameter names appear in *paramNameArray*. If you did not provide *paramNameArray*, specify the values in the order in which the stored procedure expects them.

Returns

An array of strings representing the list of columns. This function returns an error if the SQL statement is invalid or if the connection string is wrong.

Example

The following code could return a list of result set columns that were generated from the executed stored procedure, `getNewEmployeesMakingAtLeast`:

```
var paramNameArray = new Array("startDate", "salary")
var paramValueArray = new Array("2/1/2000", "50000")
var columnArray = MMDB.getSPColumnListNamedParams("EmpDB", ↵
"getNewEmployeesMakingAtLeast", paramNameArray, paramValueArray)
```

These are the returned values:

```
columnArray[0] = "EmpID", columnArray[1] = "LastName", ↵
columnArray[2] = "startDate", columnArray[3] = "salary"
```

MMDB.getSPParamsAsString()

Availability

Dreamweaver UltraDev 1.0

Description

Gets a comma-delimited string containing the list of parameters that the stored procedure takes.

Arguments

connName, *procName*

- *connName* is an UltraDev connection name as specified in the Connection Manager. It is used to make a database connection to a live data source.
- *procName* is the name of the stored procedure.

Returns

A comma-delimited string containing the list of parameters that the stored procedure requires. The parameters' names, direction, and data type are included, separated by semicolons (;).

Example

The code `MMDB.getSPParamsAsString("EmpDB", "getNewEmployeesMakingAtLeast")` could return a string of form
`name startDate;direction:in;datatype:date,`
`salary;direction:in;datatype:integer`

Here, the stored procedure, `getNewEmployeesMakingAtLeast`, has two parameters: `startDate` and `Salary`. For `startDate`, the direction is `in` and the data type is `date`. For `salary`, the direction is `in` and the data type is `date`.

MMDB.getTables()

Availability

Dreamweaver UltraDev 1.0

Description

Gets a list of all the tables defined for the specified database. Each table object has three properties: `table`, `schema`, and `catalog`. `Table` is the name of the table, `schema` is the name of the schema containing the table, and `catalog` is the catalog containing the table.

Arguments

connName

connName is an UltraDev connection name as specified in the Connection Manager. It is used to make a database connection to a live data source.

Returns

An array of objects; each object has three properties: `table`, `schema`, and `catalog`.

Example

The statement `MMDB.getTables ("EmpDB")`; might produce an array of two objects. The first object's properties might be the following:

```
object1[table:"Employees", schema:"personnel", catalog:"syscat"]
```

The second object's properties might be the following:

```
object2[table:"Departments", schema:"demo", catalog:"syscat2"]
```

MMDB.getUserName()

Availability

Dreamweaver UltraDev 1.0

Description

Returns a user name for the specified connection.

Arguments

connName

connName is an UltraDev connection name as specified in the Connection Manager. It is used to make a database connection to a live data source.

Returns

A user name string associated with the connection name.

Example

The statement `MMDB.getUserName ("EmpDB")`; might return "amit".

MMDB.getViews()

Availability

Dreamweaver UltraDev 4.0

Description

Gets a list of all the views defined for the specified database. Each view object has properties: `catalog`, `schema`, and `view`. `Catalog` or `schema` is used for restricting/filtering the number of views that pertain to an individual schema name or catalog name defined as part of the connection information.

Arguments

connName

connName is an UltraDev connection name as specified in the Connection Manager. It is used to make a database connection to a live data source.

Returns

An array of view objects; each object has three properties: `catalog`, `schema`, and `view`.

Example

The following example returns the views for a given connection value, `CONN_LIST.getValue()`:

```
var viewObjects = MMDB.getViews(CONN_LIST.getValue())
for (i = 0; i < viewObjects.length; i++)
{
    thisView = viewObjects[i]
    thisSchema = Trim(thisView.schema)
    if (thisSchema.length == 0)
    {
        thisSchema = Trim(thisView.catalog)
    }
    if (thisSchema.length > 0)
    {
        thisSchema += "."
    }
    views.push(String(thisSchema + thisView.view))
}
```

MMDB.showConnectionMgrDialog()

Availability

Dreamweaver UltraDev 1.0

Description

Displays the Connection Manager dialog box.

Arguments

Nothing.

Returns

Nothing. The Connection Manager dialog box appears.

MMDB.showResultSet()

Availability

Dreamweaver UltraDev 1.0

Description

Displays a dialog box with the results of executing the specified SQL statement. The dialog box displays a tabular grid, with the header reflecting the column information and data of the result set generated by the executed stored procedure. If the connection string or the SQL statement is invalid, an error appears. You can use this function to verify the validity of the SQL statement.

Arguments

connName, *SQLstatement*

- *connName* is an UltraDev connection name as specified in the Connection Manager. It is used to make a database connection to a live data source.
- *SQLstatement* is the SQL SELECT statement.

Returns

Nothing. This function returns an error if the SQL statement is invalid or if the connection string is wrong.

Example

The following code displays the results of the executed SQL statement:

```
MMDB.showResultSet("EmpDB","Select EmpName,EmpFirstName,Age from-  
Employees")
```

MMDB.showSPResultset()

Availability

Dreamweaver UltraDev 1.0

Description

Displays a dialog box with the results of executing the specified stored procedure. The dialog box displays a tabular grid, with the header reflecting the column information and data of the result set generated by the executed stored procedure. If the connection string or the stored procedure is invalid, an error appears. You can use this function to verify the validity of the stored procedure.

Arguments

connName, *procName*, *paramValuesArray*

- *connName* is the UltraDev connection name as specified in the Connection Manager. It is used to make a database connection to a live data source.
- *procName* is the name of the stored procedure to execute.
- *paramValuesArray* is an array containing a list of design-time parameter test values. Specify the parameter values in the order in which the stored procedure expects them. You can use the `MMDB.getSPParamsAsString()` function to get the parameters of the stored procedure.

Returns

Nothing. This function returns an error if the SQL statement is invalid or if the connection string is wrong.

Example

The following code displays the results of the executed stored procedure:

```
var paramValueArray = new Array("2/1/2000", "50000")
MMDB.showSPResultset("EmpDB", "getNewEmployeesMakingAtLeast", paramValueArray)
```

MMDB.showSPResultSetNamedParams()

Availability

Dreamweaver UltraDev 1.0

Description

Displays a dialog box with the results of executing the specified stored procedure. The dialog box displays a tabular grid, with the header reflecting the column information and data of the result set generated by the executed stored procedure. If the connection string or the stored procedure is invalid, an error appears. You can use this function to verify the validity of the stored procedure. This function differs from `MMDB.showSPResultSet()` because you can specify the parameter values by name, instead of in the order in which the stored procedure expects them.

Arguments

connName, *procName*, *paramNameArray*, *paramValuesArray*

- *connName* is an UltraDev connection name as specified in the Connection Manager. It is used to make a database connection to a live data source.
- *procName* is the name of the stored procedure that returns the result set when executed.
- *paramNameArray* is an array containing a list of parameter names. You can use the `MMDB.getSPParamsAsString()` function to get the parameters of the stored procedure.
- *paramValuesArray* is an array containing a list of design-time parameter test values.

Returns

Nothing. This function returns an error if the SQL statement is invalid or if the connection string is wrong.

Example

The following code displays the results of the executed stored procedure:

```
var paramNameArray = new Array("startDate", "salary")
var paramValueArray = new Array("2/1/2000", "50000")
MMDB.showSPResultSetNamedParams("EmpDB", "getNewEmployees-
MakingAtLeast", paramNameArray, paramValueArray)
```

CHAPTER 17

The JavaBean API

This chapter explains the APIs for JavaBeans, the `MMJB*()` functions are JavaScript hooks that invoke Java introspection calls for JavaBean support. These functions get class names, methods, properties, and events from the JavaBean, which can be displayed in your Dreamweaver user interface. To use these JavaScript functions and enable Dreamweaver to access your JavaBean, your JavaBean must reside in the Configuration/Classes folder.

Note: `packageName.className` - is one single String input.

MMJB.getProperties()

Availability

Dreamweaver UltraDev 4.0

Description

Introspects the bean class and returns its properties.

Arguments

packageName.className

packageName.className is the name of the class, which is part of the classpath. It must be a Java .jar or .zip Java archive residing in your system class path or a .class file that is installed in the Configuration/Classes folder.

Returns

A string array of the JavaBean properties. On error, returns an empty array.

MMJB.getMethods()

Availability

Dreamweaver UltraDev 4.0

Description

Introspects the bean class and returns its methods.

Arguments

packageName.className

packageName.className is the package name of the class, which is part of the classpath. It must be a Java .jar or .zip Java archive.

Returns

A string array of the JavaBean methods. On error, returns an empty array.

MMJB.getEvents()

Availability

Dreamweaver UltraDev 4.0

Description

Introspects the bean class and returns its events.

Arguments

packageName.className

packageName.className is the package name of the class, which is part of the classpath. It must be a Java .jar or .zip Java archive.

Returns

A string array of the JavaBean events. On error, returns an empty array.

MMJB.getIndexedProperties()

Availability

Dreamweaver UltraDev 4.0

Description

Introspects the bean class and returns its indexed properties. Indexed properties are design patterns that behave like collections.

Arguments

packageName.className

packageName.className is the package name of the class, which is part of the classpath. It must be a Java .jar or .zip Java archive.

Returns

A string array of indexed properties of the JavaBean. On error, returns an empty array.

MMJB.getClasses()

Availability

Dreamweaver UltraDev 4.0

Description

Reads all the JavaBean class names from the Configuration/Classes folder.

Arguments

None.

Returns

A string array of class names that live in Configuration/Classes folder. On error, returns an empty array.

MMJB.getClassesFromPackage()

Availability

Dreamweaver UltraDev 4.0

Description

Reads all the JavaBean classes from the package.

Arguments

packageName.pathName

packageName.pathName is the path to the package. It must be a Java .jar or .zip Java archive. For example, C:\jdbcdrivers\Una2000_Enterprise.zip.

Returns

A string array of class names inside the particular .jar or .zip Java file. On error, returns an empty array.

MMJB.getErrorMessage()

Availability

Dreamweaver UltraDev 4.0

Description

Gets the last error message from Dreamweaver that occurred while using the MMJB interface.

Arguments

None.

Returns

A string of the Dreamweaver message from the last error.

CHAPTER 18

The Source Control Integration API

The source control integration API allows you to write DLLs to extend the Dreamweaver check in/check out feature using the features of other source control systems (such as Sourcesafe, CVS, or Whirlwind). These API functions can also be used to describe how the Dreamweaver user can interact with the integrated source control system.

You implement this API by writing a C-level DLL/shared code fragment that adheres to the API defined in this chapter. There is a minimum set of API functions that you must support in order for Dreamweaver to integrate with a source control system. The DLLs are placed in the Configuration/SourceControl folder.

When started, Dreamweaver loads each DLL file. Dreamweaver determines which of the APIs the DLL supports by calling `GetProcAddress()` on the various APIs. If an address doesn't exist, Dreamweaver assumes the DLL does not support the API. If the address exists, Dreamweaver uses the DLL's version of the function to support the functionality. When a Dreamweaver user defines or edits a site and then chooses the Web Server SCS tab, the choices corresponding to the DLLs that were loaded from the Configuration/SourceControl folder appear (in addition to the standard items) on the tab.

To add custom items to the Site > Source Control menu, add the following code in the Site menu in the `menus.xml` file:

```
<menu name="Source Control" id="DWMenu_MainSite_Site_Source-
Control"><menuitem dynamic name="None" file="Menus/MM/
File_SCSItems.htm" id="DWMenu_MainSite_Site_NewFeatures_Default" />
</menu>
```

Integration with Dreamweaver

When a Dreamweaver user chooses server connection, file transfer, or Design Notes features, Dreamweaver calls the DLL's version of the corresponding API function (`Connect()`, `Disconnect()`, `Get()`, `Put()`, `Checkin()`, `Checkout()`, `Undocheckout()`, and `Synchronize()`). The DLL is responsible for handling the request, including displaying dialog boxes that gather information or allow the user to interact with the DLL. The DLL is also responsible for displaying information or error messages.

The source control system can optionally support Design Notes and Check In/Check Out. The Dreamweaver user enables Design Notes in source control systems by choosing the Design Notes tab in the Define Sites dialog and checking the box to enable the feature; this is the same way that they enable Design Notes with FTP and LAN in version 3.0. If the source control system does not support Design Notes and the user wants to use this feature, Dreamweaver transports Design Note (.mno) files to maintain the Design Notes (as it does with FTP and LAN in version 3.0).

Check In/Check Out is treated differently than the Design Notes feature; if the source control system supports it, the user cannot override its use from the Design Notes dialog box. If the user tries to override the source control system, an error message appears.

Adding source control system functionality

You can add source control system functionality to Dreamweaver by writing a `GetNewFeatures` handler that returns a set of menu items and corresponding C functions. For example, if you were writing a Sourcesafe DLL and wanted to allow Dreamweaver users to see the history of a file, you could write a `GetNewFeatures` handler that returned the History menu item and the C function name of `history`. Then, when the user right-clicks a file, the History menu item is one of the items on the menu. If a user chooses the History menu item, Dreamweaver calls the history function, passing the selected file(s) to the DLL. You would then have the DLL display the History dialog box so the user could interact with it just like they would if using Sourcesafe.

The source control integration API required functions

The source control integration API has both required and optional functions. The functions listed in this section are required.

SCS_GetAgentInfo()

Description

Asks the DLL to return its name and description, which are displayed in the Define Sites dialog. Name appears in the Server Access pop-up menu (for example, sourcesafe, webdav, perforce) and description just below the pop-up menu.

Arguments

name, version, description, dwAppVersion

- *name* is the name of the source control system. Name appears in the combo box for selecting a source control system in the Source Control tab of the Define Sites dialog box. The name can be a maximum of 32 characters.
- *version* is a string indicating the version of the DLL. Version appears in the Source Control tab of the Define Sites dialog. The version can be a maximum of 32 characters.
- *description* is a string indicating the description of the source control system. Description appears in the Source Control tab of the Define Sites dialog. The description can be a maximum of 256 characters.
- *dwAppVersion* is a string representing the version of Dreamweaver that is calling the DLL. The DLL can use this string to determine the version and language of Dreamweaver.

Returns

true if successful, false if not.

SCS_Connect()

Description

Connects the user to the source control system. If the DLL does not have login information, the DLL is responsible for displaying a dialog box to prompt the user for the information and for storing the data for later use.

Arguments

connectionData, siteName

- *connectionData* is a handle to the data the agent wants Dreamweaver to pass to it when calling other API functions.
- *siteName* is a string pointing to the name of the site. The site name can be a maximum of 64 characters.

Returns

true if successful, false if not.

SCS_Disconnect()

Description

Disconnects the user from the source control system.

Arguments

connectionData

connectionData is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.

Returns

true if successful, false if not.

SCS_IsConnected()

Description

Determines the state of the connection.

Arguments

connectionData

connectionData is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.

Returns

true if connected, false if not.

SCS_GetRootFolderLength()

Description

Returns the length of the name of the root folder.

Arguments

connectionData

connectionData is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.

Returns

An integer representing the length of the name of the root folder. If the function returns < 0 , Dreamweaver considers it an error and tries to retrieve the error message from the DLL if supported.

SCS_GetRootFolder()

Description

Returns the name of the root folder.

Arguments

connectionData, *remotePath*, *folderLen*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePath* is a buffer where the full remote path of the root folder is stored.
- *folderLen* is an integer indicating the length of *remotePath*. This is the value returned from *GetRootFolderLength*.

Returns

true if successful, false if not.

SCS_GetFolderListLength()

Description

Returns the number of items in the passed-in folder.

Arguments

connectionData, *remotePath*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePath* is the fully qualified path of the remote folder that the DLL checks for the number of items.

Returns

An integer indicating the number of items in the current folder. If the function returns < 0 , Dreamweaver considers it an error and tries to retrieve the error message from the DLL, if supported.

SCS_GetFolderList()

Description

Returns a list of files and folders in the passed-in folder, including pertinent information such as modified date, size, and whether the item is a folder or file.

Arguments

connectionData, *remotePath*, *itemList*, *numItems*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePath* is the fully qualified path of the remote folder that the DLL checks for the number of items.
- *itemList* is a preallocated list of `itemInfo` structures.
- *numItems* is the number of items allocated for the *itemList* (returned from `GetFolderListLength`).

Returns

true if successful, false if not.

SCS_Get()

Description

Gets a list of files or folders and stores them locally.

Arguments

connectionData, *remotePathList*, *localPathList*, *numItems*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePathList* is a list of fully qualified remote file or folder paths to get.
- *localPathList* is a mirrored list of fully qualified local file or folder paths.
- *numItems* is the number of items in each list.

Returns

true if successful, false if not.

SCS_Put()

Description

Puts a list of local files or folders into the source control system.

Arguments

connectionData, *localPathList*, *remotePathList*, *numItems*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *localPathList* is a list of fully qualified local file or folder paths to put.
- *remotePathList* is a mirrored list of fully qualified remote file or folder paths.
- *numItems* is the number of items in each list.

Returns

true if successful, false if not.

SCS_NewFolder()

Description

Creates a new folder.

Arguments

connectionData, *remotePath*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePath* is the fully qualified name of the remote folder the DLL creates.

Returns

true if successful, false if not.

SCS_Delete()

Description

Deletes a list of files or folders from the source control system.

Arguments

connectionData, *remotePathList*, *numItems*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePathList* is a list of fully qualified remote file or folder paths to delete.
- *numItems* is the number of items in *remotePathList*.

Returns

true if successful, false if not.

SCS_Rename()

Description

Renames or moves a file or folder depending on the values specified for *oldRemotePath* and *newRemotePath*. For example, if *oldRemotePath* equals `"/folder1/file1"` and *newRemotePath* equals `"/folder1/renamefile1"`, file1 is renamed to renamefile1 and is located in folder1.

If *oldRemotePath* equals `"/folder1/file1"` and *newRemotePath* equals `"/folder1/subfolder1/file1"`, file1 is moved to the subfolder1 directory.

To find out if an invocation of this function is a move or a rename, check the parent paths of the two input values; if they are the same, the operation is a rename.

Arguments

connectionData, *oldRemotePath*, *newRemotePath*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *oldRemotePath* is a fully qualified remote file or folder path to rename.
- *newRemotePath* is the fully qualified remote path of the new name for the file or folder.

Returns

true if successful, false if not.

SCS_ItemExists()

Description

Determines whether or not a file or folder exists on the server.

Arguments

connectionData, *remotePath*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePath* is a fully qualified remote file or folder path.

Returns

true if exists, false if not.

The source control integration API optional functions

The source control integration API has both required and optional functions. The functions in this section are optional.

SCS_GetConnectionInfo()

Description

Displays a dialog box to allow the user to change or set the connection information for this site. Does not make the connection. This is called when the user clicks the “Settings...” button in the Remote Info section of the Define Sites dialog box.

Arguments

connectionData, *siteName*

- *connectionData* is a handle to data that the agent wants Dreamweaver to pass it when calling other API functions.
- *siteName* is a string pointing to the name of the site, which cannot exceed 64 characters.

Returns

true if successful, false if not.

SCS_SiteDeleted()

Description

Notifies the DLL that the site has been deleted or that the site is no longer tied to this source control system, indicating that the source control system can delete its persistent information for this site.

Arguments

siteName

siteName is a string pointing to the name of the site, which cannot exceed 64 characters.

Returns

true if successful, false if not.

SCS_SiteRenamed()

Description

Notifies the DLL when the user has renamed the site so that it can update its persistent information about the site.

Arguments

oldSiteName, *newSiteName*

- *oldSiteName* is a string pointing to the original name of the site before it was renamed, which cannot exceed 64 characters.
- *newSiteName* is a string pointing to the new name of the site after it was renamed, which cannot exceed 64 characters.

Returns

true if successful, false if not.

SCS_GetNumNewFeatures()

Description

Returns the number of new features to add to Dreamweaver (for example, File History, Differences, and so on).

Arguments

None.

Returns

An integer representing the number of new features to add to Dreamweaver. If the function returns < 0, Dreamweaver considers it an error and tries to retrieve the error message from the DLL, if supported.

SCS_GetNewFeatures()

Description

Returns a list of menu items to add to the Dreamweaver main and context menus. For example, the Sourcesafe DLL could add History and File Differences to the main menu.

Arguments

menuItemList, *menuItemList*, *enablerList*, *numNewFeatures*

- *menuItemList* is a string list populated by the DLL; it specifies the menu items to add to the main and context menus. Each string can contain a maximum of 32 characters.
- *functionList* is populated by the DLL; it specifies the routines in the DLL to call when the user chooses the corresponding menu item.
- *enablerList* is populated by the DLL; it specifies the routines in the DLL to call when Dreamweaver needs to determine whether the corresponding menu item is enabled.
- *numNewFeatures* is the number of items being added by the DLL; this is retrieved from the `GetNumNewFeatures()` call.

Returns

true if successful, false if not.

SCS_GetCheckoutName()

Description

Returns the checkout name of the current user. If unsupported by the source control system and this feature is enabled by the user, uses the Dreamweaver internal checkin/out functionality, which transports .lck files to and from the source control system.

Arguments

connectionData, *checkOutName*, *emailAddress*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *checkOutName* is the checkout name of the current user.
- *emailAddress* is the email address of the current user.

Returns

true if successful, false if not.

SCS_Checkin()

Description

Checks a list of local files or folders into the source control system. The DLL is responsible for making the file read-only. If unsupported by the source control system and this feature is enabled by the user, uses the Dreamweaver internal checkin/out functionality, which transports .lck files to and from the source control system.

Arguments

connectionData, *localPathList*, *remotePathList*, *successList*, *numItems*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *localPathList* is a list of fully qualified local file or folder paths to check in.
- *remotePathList* is a mirrored list of fully qualified remote file or folder paths.
- *successList* is a list of Boolean values populated by the DLL to let Dreamweaver know which of the corresponding files was successfully checked in.
- *numItems* is the number of items in each list.

Returns

true if successful, false if not.

SCS_Checkout()

Description

Checks out a list of local files or folders from the source control system. The DLL is responsible for granting the privileges that allow the file to be writeable. If unsupported by the source control system and this feature is enabled by the user, uses the Dreamweaver internal checkin/out functionality, which transports .lck files to/from the source control system.

Arguments

connectionData, *remotePathList*, *localPathList*, *successList*, *numItems*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePathList* is a list of fully qualified remote file or folder paths to check out.
- *localPathList* is a mirrored list of fully qualified local file or folder paths.
- *successList* is a list of Boolean values populated by the DLL to let Dreamweaver know which of the corresponding files was successfully checked out.
- *numItems* is the number of items in each list.

Returns

true if successful, false if not.

SCS_UndoCheckout()

Description

Undoes the checkout status of a list of files or folders. The DLL is responsible for making the file read-only. If unsupported by the source control system and this feature is enabled by the user, uses the Dreamweaver internal checkin/out functionality, which transports .lck files to/from the source control system.

Arguments

connectionData, *remotePathList*, *localPathList*, *successList*, *numItems*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePathList* is a list of fully qualified remote file or folder paths on which to undo the check out.
- *localPathList* is a mirrored list of fully qualified local file or folder paths.
- *successList* is a list of Boolean values populated by the DLL to let Dreamweaver know which corresponding files' check outs were successfully undone.
- *numItems* is the number of items in each list.

Returns

true if successful, false if not.

SCS_GetNumCheckedOut()

Description

Returns the number of people who have a file checked out.

Arguments

connectionData, *remotePath*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePath* is the fully qualified remote file or folder path to check to see how many users have it checked out.

Returns

An integer representing the number of people who have the file checked out. If the function returns < 0 , Dreamweaver considers it an error and tries to retrieve the error message from the DLL if supported.

SCS_GetFileCheckoutList()

Description

Returns a list of people who have a file checked out. If the list is empty, no one has the file checked out.

Arguments

connectionData, *remotePath*, *checkOutList*, *emailAddressList*,
numCheckedOut

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePath* is the fully qualified remote file or folder path to check to see how many users have it checked out.
- *checkOutList* is a list of strings corresponding to the users who have the file checked out. Each user string cannot exceed a maximum length of 64 characters.
- *emailAddressList* is a list of strings corresponding to the users' email addresses. Each email address string cannot exceed a maximum length of 64 characters.
- *numCheckedOut* is the number of people who have the file checked out. This is returned from `GetNumCheckedOut()`.

Returns

true if successful, false if not.

SCS_GetErrorMessageLength()

Description

Returns the length of the DLL's current internal error message. This is used to allocate the buffer passed into the `GetErrorMessage()` function. This function should be called only if an API function returns `false` or `<0`, indicating a failure of that API function.

Arguments

connectionData

connectionData is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.

Returns

An integer representing the length of the error message.

SCS_GetErrorMessage()

Description

Returns the last error message. If you implement `getErrorMessage()`, Dreamweaver calls it each time one of your API functions returns `false`.

If a routine returns `-1` or `false`, it indicates an error message should be available.

Arguments

connectionData, *errorMsg*, *msgLength*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *errorMsg* is a preallocated string for the DLL to fill in with the error message.
- *msgLength* is the length of the *errorMsg* buffer passed in.

Returns

true if successful, false if not.

SCS_GetNoteCount()

Description

Returns the number of Design Note keys for the specified remote file or folder path. If unsupported by the source control system, Dreamweaver gets this information from the companion Design Note (.mno) file.

Arguments

connectionData, *remotePath*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePath* is the fully qualified remote file or folder path that the DLL checks for the number of Design Notes attached to it.

Returns

An integer representing the number of Design Notes associated with this file. If the function returns < 0 , Dreamweaver considers it an error and tries to retrieve the error message from the DLL, if supported.

SCS_GetMaxNoteLength()

Description

Returns the length of the largest Design Note for the specified file or folder. If unsupported by the source control system, Dreamweaver gets this information from the companion Design Note (.mno) file.

Arguments

connectionData, *remotePath*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePath* is the fully qualified remote file or folder path that the DLL checks for the maximum Design Note length.

Returns

An integer representing the size of the longest Design Note associated with this file. If the function returns < 0 , Dreamweaver considers it an error and tries to retrieve the error message from the DLL, if supported.

SCS_GetDesignNotes()

Description

Retrieves key-value pairs from the meta information for the specified file or folder. If unsupported by the source control system, Dreamweaver retrieves the information from the corresponding Design Note (.mno) file.

Arguments

connectionData, *remotePath*, *keyList*, *valueList*, *showColumnList*, *noteCount*, *noteLength*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePath* is the fully qualified remote file or folder path that the DLL checks for the number of items.
- *keyList* is a list of Design Note keys, such as "Status".
- *valueList* is a list of Design Note values corresponding to the Design Note keys, such as "Awaiting Signoff".
- *showColumnList* is a list of Boolean values corresponding to the Design Note keys, which indicate whether Dreamweaver can display the key as a column in the Site window.
- *noteCount* is the number of Design Notes attached to this file or folder; this value is returned by the `GetNoteCount()` call.
- *noteLength* is the maximum length of a Design Note; this is the value returned by the `GetMaxNoteLength()` call.

Returns

true if successful, false if not.

SCS_SetDesignNotes()

Description

Stores the key-value pairs in the meta information for the specified file or folder. This replaces the set of meta information for the file. If unsupported by the source control system, Dreamweaver stores Design Notes in .mno files.

Arguments

connectionData, *remotePath*, *keyList*, *valueList*, *showColumnList*, *noteCount*, *noteLength*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePath* is the fully qualified remote file or folder path that the DLL checks for the number of items.
- *keyList* is a list of Design Note keys, such as "Status".
- *valueList* is a list of Design Note values corresponding to the Design Note keys, such as "Awaiting Signoff".
- *showColumnList* is a list of Boolean values corresponding to the Design Note keys, which indicate whether Dreamweaver can display the key as a column in the Site window.
- *noteCount* is the number of Design Notes attached to this file or folder; this lets the DLL know how big the specified lists are. If *noteCount* is 0, all Design Notes are removed from the file.
- *noteLength* is the length of the largest Design note for the specified file or folder.

Returns

true if successful, false if not.

SCS_IsRemoteNewer()

Description

Checks each specified remote path to see if the remote copy is newer. If unsupported by the source control system, Dreamweaver uses its internal `isRemoteNewer` algorithm.

Arguments

connectionData, *remotePathList*, *localPathList*, *remoteIsNewerList*, *numItems*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePathList* is a list of fully qualified remote file or folder paths to compare for newer status.
- *localPathList* is a mirrored list of fully qualified local file or folder paths.
- *remoteIsNewerList* is a list of integers, populated by the DLL to let Dreamweaver know which of the corresponding files is newer on the remote side. The following values are valid: 1 indicates the remote version is newer, -1 indicates the local version is newer, 0 indicates the versions are the same.
- *numItems* is the number of items in each list.

Returns

true if successful, false if not.

Enablers

If the optional enablers are not supported by the source control system or the application is not connected to the server, then Dreamweaver determines when the menu items are enabled based on the information it has about the remote files.

SCS_canConnect()

Description

Returns whether or not the Connect menu item should be enabled.

Arguments

None.

Returns

true if enabled, false if not.

SCS_canGet()

Description

Returns whether or not the Get menu item should be enabled.

Arguments

connectionData, *remotePathList*, *localPathList*, *numItems*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePathList* is a list of fully qualified remote file or folder paths to get.
- *localPathList* is a mirrored list of fully qualified local file or folder paths.
- *numItems* is the number of items in each list.

Returns

true if enabled, false if not.

SCS_canCheckout()

Description

Returns whether or not the Checkout menu item should be enabled.

Arguments

connectionData, *remotePathList*, *localPathList*, *numItems*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePathList* is a list of fully qualified remote file or folder paths to check out.
- *localPathList* is a mirrored list of fully qualified local file or folder paths.
- *numItems* is the number of items in each list.

Returns

true if enabled, false if not.

SCS_canPut()

Description

Returns whether or not the Put menu item should be enabled.

Arguments

connectionData, *localPathList*, *remotePathList*, *numItems*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *localPathList* is a list of fully qualified local file or folder paths to put.
- *remotePathList* is a mirrored list of fully qualified remote file or folder paths.
- *numItems* is the number of items in each list.

Returns

true if enabled, false if not.

SCS_canCheckin()

Description

Returns whether or not the Checkin menu item should be enabled.

Arguments

connectionData, localPathList, remotePathList, numItems

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *localPathList* is a list of fully qualified local file or folder paths to check in.
- *remotePathList* is a mirrored list of fully qualified remote file or folder paths.
- *numItems* is the number of items in each list.

Returns

true if enabled, false if not.

SCS_CanUndoCheckout()

Description

Returns whether or not the Undo Checkout menu item should be enabled.

Arguments

connectionData, remotePathList, localPathList, numItems

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePathList* is a list of fully qualified remote file or folder paths to check out.
- *localPathList* is a list of fully qualified local file or folder paths to put.
- *numItems* is the number of items in each list.

Returns

true if enabled, false if not.

SCS_canNewFolder()

Description

Returns whether or not the New Folder menu item should be enabled.

Arguments

connectionData, *remotePath*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePath* is a fully qualified remote file or folder path that the user has selected to indicate where the new folder will be created.

Returns

true if enabled, false if not.

SCS_canDelete()

Description

Returns whether or not the Delete menu item should be enabled.

Arguments

connectionData, *remotePathList*, *numItems*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePathList* is a list of fully qualified remote file or folder paths to delete.
- *numItems* is the number of items in each list.

Returns

true if enabled, false if not.

SCS_canRename()

Description

Returns whether or not the Rename menu item should be enabled.

Arguments

connectionData, *remotePathList*

- *connectionData* is a pointer to the agent's data that was passed into Dreamweaver during the `Connect()` call.
- *remotePath* is the fully qualified remote file or folder path that could be renamed.

Returns

true if enabled, false if not.

itemInfo struct

name	char[256]	name of file or folder
isFolder	bool	true if folder, false if file
month	int	month component of mod date 1-12
day	int	day component of mod date 1-31
year	int	year component of mod date 1900+
hour	int	hour component of mod date 0-23
minutes	int	minute component of mod date 0-59
seconds	int	second component of mod date 0-59
type	char[256]	type of file (if not set by DLL, DW will use file extension to determine type, as it does now)
size	int	in bytes

SCS_BeforeGet()

Description

Dreamweaver calls this function before getting or checking out one or more files. This enables your DLL to perform one operation, such as adding a checkout comment, to a group of files.

Arguments

**connectionData*

**connectionData* is a pointer to the connection data.

Returns

A Boolean true if successful, false if unsuccessful.

Example

To get a group of files, Dreamweaver makes calls to the DLL in the following order:

```
SCS_BeforeGet(connectionData);
SCS_Get(connectionData,remotePathList1,localPathList1,successList1);
SCS_Get(connectionData,remotePathList2,localPathList2,successList2);
SCS_Get(connectionData,remotePathList3,localPathList3,successList3);
SCS_AfterGet(connectionData);
```


SCS_BeforePut()

Description

Dreamweaver calls this function before putting or checking in one or more files. This enables your DLL to perform one operation, such as adding a checkin comment, to a group of files.

Arguments

**connectionData*

**connectionData* is a pointer to the connection data.

Returns

A Boolean true if successful, false if unsuccessful.

Example

To get a group of files, Dreamweaver makes calls to the DLL in the following order:

```
SCS_BeforePut(connectionData);
SCS_Put(connectionData,localPathList1,remotePathList1,successList1);
SCS_Put(connectionData,localPathList2,remotePathList2,successList2);
SCS_Put(connectionData,localPathList3,remotePathList3,successList3);
SCS_AfterPut(connectionData);
```

SCS_AfterGet()

Description

Dreamweaver calls this function after getting or checking out one or more files. This enables your DLL to perform any operation after a batch get or checkout, such as putting up a summary dialog box.

Arguments

**connectionData*

**connectionData* is a pointer to the connection data.

Returns

A Boolean true if successful, false if unsuccessful.

Example

See example in “SCS_BeforeGet()” on page 188.

SCS_AfterPut()

Description

Dreamweaver calls this function after putting or checking in one or more files. This enables the DLL to perform any operation after a batch put or checkin, such as putting up a summary dialog box.

Arguments

**connectionData*

**connectionData* is a pointer to the connection data.

Returns

true if successful, false if unsuccessful.

Example

See example in “SCS_BeforePut()” on page 189.

CHAPTER 19

C-Level Extensibility

The C-level extensibility mechanism lets you implement Dreamweaver extensibility files using a combination of JavaScript and your own C code. You define functions using C, bundle them in a DLL or shared library, save the library in the Configuration/JSExtensions folder within the Dreamweaver application folder, and then call the functions from JavaScript using the JavaScript interpreter built into Dreamweaver.

For example, you may want to define a Dreamweaver object that inserts the contents of a user-specified file into the current document. Because client-side JavaScript does not provide support for file I/O, you must write a function in C to provide this functionality.

You could use the following HTML and JavaScript to create a simple Insert Text from File object. Notice that the `objectTag()` function calls a C function named `readContentsOfFile()`, which is stored in a library named `myLibrary`.

```
<HTML>
<HEAD>
<SCRIPT>
function objectTag() {
    fileName = document.forms[0].myFile.value;
    return myLibrary.readContentsOfFile(fileName);
}
</SCRIPT>
</HEAD>

<BODY>
<FORM>
Enter the name of the file to be inserted:
<INPUT TYPE="file" NAME="myFile">
</FORM>
</BODY>
</HTML>
```

The `readContentsOfFile()` function accepts a list of arguments from the user, unpacks the argument containing the file name, reads the contents of the file, and packages the contents of the file as the return value. For more information about the JavaScript data structures and functions that appear in `readContentsOfFile()`, see “The C-level extensibility API” on page 193.

```
JSBool
readContentsOfFile(JSContext *cx, JSObject *obj, unsigned int argc,
jsval *argv, jsval *rval)
{
    char *fileName, *fileContents;
    JSBool success;
    unsigned int length;

    /* Make sure caller passed in exactly one argument. If not,
     * then tell the interpreter to abort script execution. */
    if (argc != 1){
        JS_ReportError(cx, "Wrong number of arguments", 0);
        return JS_FALSE;
    }

    /* Convert the argument to a string */
    fileName = JS_ValueToString(cx, argv[0], &length);
    if (fileName == NULL){
        JS_ReportError(cx, "The argument must be a string", 0);
        return JS_FALSE;
    }

    /* Use the string (the file name) to open and read a file */
    fileContents = exerciseLeftToTheReader(fileName);

    /* Store file contents in rval, which is the return value
     * passed
     * back to the caller */
    success = JS_StringToValue(cx, fileContents, 0, *rval);
    free(fileContents);

    /* Return true to continue or false to abort the script */
    return success;
}
```

To ensure that the `readContentsOfFile()` function executes as designed rather than causing a JavaScript error, you must register the function with the JavaScript interpreter by including a function called `MM_Init()` in your library. When Dreamweaver loads the library at startup, it calls the `MM_Init()` function to get three pieces of information:

- The JavaScript name of the function
- A pointer to the function
- The number of arguments that the function expects

The `MM_Init()` function for `myLibrary` might look like this:

```
void
MM_Init()
{
    JS_DefineFunction("readContentsOfFile", readContentsOfFile,
1);
}
```

Your library must include exactly one instance of the following macro:

```
/* MM_STATE is a macro that expands to some definitions that are
 * needed to interact with Dreamweaver. This macro must
 * be defined exactly once in your library. */
MM_STATE
```

Note: The library can be implemented in either C or C++, but the file containing `MM_Init()` and `MM_STATE` must be implemented in C. The C++ compiler garbles function names, making it impossible for Dreamweaver to find the `MM_Init()` function.

The C-level extensibility API

The C code in your library must interact with the Dreamweaver JavaScript interpreter at three different times:

- At startup, to register the library's functions.
- When the function is called, to unpack the arguments that are being passed from JavaScript to C.
- Before the function returns, to package the return value.

To accomplish these tasks, the interpreter defines several data types and exposes an API. Definitions for the data types and functions listed in this section appear in the file `mm_jsapi.h`. For your library to work properly, you must include `mm_jsapi.h` at the top of each file in your library with the following line:

```
#include "mm_jsapi.h"
```

typedef struct JSContext JSContext

Description

A pointer to this opaque data type is passed to the C-level function. Some of the functions in the API accept this pointer as one of their arguments.

typedef struct JSObject JSObject

Description

A pointer to this opaque data type is passed to the C-level function. This data type represents an object, which may be an array object or some other object type.

typedef struct jsval jsval

Description

An opaque data structure that can contain an integer, or a pointer to a float, string, or object. Some functions in the API can be used to read the values of function arguments by reading the contents of a `jsval`, and some can be used to write the function's return value by writing a `jsval`.

typedef enum { JS_FALSE = 0, JS_TRUE = 1 } JSBool

Description

A simple data type used to store a Boolean value.

typedef JSBool (*JSNative)(JSContext *cx, JSObject *obj, unsigned int argc, jsval *argv, jsval *rval)

Description

The function signature for C-level implementations of JavaScript functions, where:

- *cx* is a pointer to an opaque `JSContext` structure, which must be passed to some of the functions in the JavaScript API. This variable holds the interpreter's execution context.
- *obj* is a pointer to the object in whose context the script executes. While the script is running, the `this` keyword is equal to this object.
- *argc* is the number of arguments being passed to the function.
- *argv* is a pointer to an array of `jsvals`. The array is *argc* elements in length.
- *rval* is a pointer to a single `jsval`. The function's return value should be written to **rval*.

The function returns `JS_TRUE` upon success or `JS_FALSE` upon failure. If `JS_FALSE` is returned, the current script stops executing.

JSBool JS_DefineFunction()

Description

Registers a C-level function with the JavaScript interpreter in Dreamweaver. After this function returns, JavaScript scripts that call the function specified in *name* will execute the code pointed to by *call*.

Typically, this function is called from `MM_Init()`, which Dreamweaver calls during startup.

Arguments

char **name*, JSNative *call*, unsigned int *nargs*

- *name* is the name of the function as it is exposed to JavaScript.
- *call* is a pointer to a C-level function. The function must accept the same arguments as `readContentsOfFile`, and it must return a `JSBool` indicating success or failure.
- *nargs* is the number of arguments that the function expects to receive.

Returns

A Boolean value indicating success (`JS_TRUE`) or failure (`JS_FALSE`).

char *JS_ValueToString()

Description

Extracts a function argument from a `jsval`, converts it to a string (if possible), and passes the converted value back to the caller.

Arguments

JSContext **cx*, jsval *v*, unsigned int **pLength*

- *cx* is the opaque `JSContext` pointer that was passed to the JavaScript function.
- *v* is the `jsval` from which the string is to be extracted.
- *pLength* is a pointer to an unsigned integer. This function sets **pLength* equal to the length of the string in bytes.

Returns

A pointer to a string on success or `null` on failure.

JSBool JS_ValueToInteger()

Description

Extracts a function argument from a `jsval`, converts it to an integer (if possible), and passes the converted value back to the caller.

Arguments

JSContext **cx*, jsval *v*, long **lp*

- *cx* is the opaque `JSContext` pointer that was passed to the JavaScript function.
- *v* is the `jsval` from which the string is to be extracted.
- *lp* is a pointer to a 4-byte-long integer. This function stores the converted value in **lp*.

Returns

A Boolean value indicating success (`JS_TRUE`) or failure (`JS_FALSE`).

JSBool JS_ValueToDouble()

Description

Extracts a function argument from a `jsval`, converts it to a double (if possible), and passes the converted value back to the caller.

Arguments

`JSContext *cx`, `jsval v`, `double *dp`

- `cx` is the opaque `JSContext` pointer that was passed to the JavaScript function.
- `v` is the `jsval` from which the string is to be extracted.
- `dp` is a pointer to an 8-byte double. This function stores the converted value in `*dp`.

Returns

A Boolean value indicating success (`JS_TRUE`) or failure (`JS_FALSE`).

JSBool JS_ValueToBoolean()

Description

Extracts a function argument from a `jsval`, converts it to a Boolean value (if possible), and passes the converted value back to the caller.

Arguments

`JSContext *cx`, `jsval v`, `JSBool *bp`

- `cx` is the opaque `JSContext` pointer that was passed to the JavaScript function.
- `v` is the `jsval` from which the string is to be extracted.
- `bp` is a pointer to a `JSBool`. This function stores the converted value in `*bp`.

Returns

A Boolean value indicating success (`JS_TRUE`) or failure (`JS_FALSE`).

JSBool JS_ValueToObject()

Description

Extracts a function argument from a `jsval`, converts it to an object (if possible), and passes the converted value back to the caller. If the object is an array, use `JS_GetArrayLength()` and `JS_GetElement()` to read its contents.

Arguments

`JSContext *cx`, `jsval v`, `JSObject **op`

- *cx* is the opaque `JSContext` pointer that was passed to the JavaScript function.
- *v* is the `jsval` from which the string is to be extracted.
- *op* is a pointer to a (`JSObject *`). This function stores the converted value in **op*.

Returns

A Boolean value indicating success (`JS_TRUE`) or failure (`JS_FALSE`).

JSBool JS_StringToValue()**Description**

Stores a string return value in a `jsval`.

Arguments

`JSContext *cx`, `char *bytes`, `size_t sz`, `jsval *vp`

- *cx* is the opaque `JSContext` pointer that was passed to the JavaScript function.
- *bytes* is the string to be stored in the `jsval`. The string data is copied, so the caller should free the string when it is no longer needed. If the string size is not specified (see the *sz* argument), then the string must be null-terminated.
- *sz* is the size of the string, in bytes. If *sz* is 0, then the length of the null-terminated string is computed automatically.
- *vp* is a pointer to the `jsval` into which the contents of the string should be copied.

Returns

A Boolean value indicating success (`JS_TRUE`) or failure (`JS_FALSE`).

JSBool JS_DoubleToValue()**Description**

Stores a floating-point number return value in a `jsval`.

Arguments

`JSContext *cx`, `double dv`, `jsval *vp`

- *cx* is the opaque `JSContext` pointer that was passed to the JavaScript function.
- *dv* is an 8-byte floating-point number.
- *vp* is a pointer to the `jsval` into which the contents of the double should be copied.

Returns

A Boolean value indicating success (`JS_TRUE`) or failure (`JS_FALSE`).

JSVal JS_BooleanToValue()

Description

Stores a Boolean return value in a `jsval`.

Arguments

JSBool *bv*

Returns

A JSVal containing the Boolean value passed in.

JSVal JS_IntegerToValue()

Description

Stores an integer return value in a `jsval`.

Arguments

long *lv*

Returns

A JSVal containing the integer passed in.

JSVal JS_ObjectToValue()

Description

Stores an object return value in a `jsval`. Use `JS_NewArrayObject()` to create an array object; use `JS_SetElement()` to define its contents.

Arguments

JSObject **obj*

Returns

A JSVal containing the object passed in.

char *JS_ObjectType()

Description

Given an object reference, returns a string describing the type of the object. For array objects, the return value is `Array`.

Arguments

JSObject **obj*

Typically, this argument is passed in and converted using `JS_ValueToObject()`.

Returns

A pointer to a null-terminated string. The caller should *not* free this string when it has finished.

JSObject *JS_NewArrayObject()

Description

Creates a new object that contains an array of `jsvals`.

Arguments

`JSContext *cx`, unsigned int `length`, `jsval *v`

- `cx` is the opaque `JSContext` pointer that was passed to the JavaScript function.
- `length` is the number of elements that the array will hold.
- `v` is an optional pointer to the `jsvals` to be stored in the array. If the return value is not `null`, `v` is an array containing `length` elements. If the return value is `null`, then the initial content of the array object is undefined (and may be set using `JS_SetElement()`).

Returns

A pointer to a new array object, or `null` upon failure.

long JS_GetArrayLength()

Description

Given a pointer to an array object, gets the number of elements in the array.

Arguments

`JSContext *cx`, `JSObject *obj`

- `cx` is the opaque `JSContext` pointer that was passed to the JavaScript function.
- `obj` is a reference to an array object.

Returns

The number of elements in the array, or -1 upon failure.

JSBool JS_GetElement()

Description

Reads a single element of an array object.

Arguments

JSContext **cx*, JSObject **obj*, unsigned int *index*, jsval **v*

- *cx* is the opaque JSContext pointer that was passed to the JavaScript function.
- *obj* is a pointer to an array object.
- *index* is an integer index into the array. The first element is index 0, and the last element is index (length - 1).
- *v* is a pointer to a jsval where the contents of the jsval in the array should be copied.

Returns

A Boolean value indicating success (JS_TRUE) or failure (JS_FALSE).

JSBool JS_SetElement()

Description

Writes a single element of an array object.

Arguments

JSContext **cx*, JSObject **obj*, unsigned int *index*, jsval **v*

- *cx* is the opaque JSContext pointer that was passed to the JavaScript function.
- *obj* is a pointer to an array object.
- *index* is an integer index into the array. The first element is index 0, and the last element is index (length - 1).
- *v* is a pointer to a jsval whose contents should be copied to the jsval in the array.

Returns

A Boolean value indicating success (JS_TRUE) or failure (JS_FALSE).

JSBool JS_ExecuteScript()

Description

Compiles and executes a string of JavaScript. If the script generates a return value, it is returned in **rval*.

Arguments

*JSContext *cx*, *JSObject *obj*, *char *script*, *unsigned in sz*,
*jsval *rval*

- *cx* is the opaque *JSContext* pointer that was passed to the JavaScript function.
- *obj* is a pointer to the object in whose context the script executes. While the script is running, the *this* keyword is equal to this object. Usually this is the *JSObject* pointer that was passed to the JavaScript function.
- *script* is a string containing JavaScript code. If the string size is not specified (see the *sz* argument), then the string must be null-terminated.
- *sz* is the size of the string, in bytes. If *sz* is 0, then the length of the null-terminated string is computed automatically.
- *rval* is a pointer to a single *jsval*. The function's return value is stored in **rval*.

Returns

A Boolean value indicating success (*JS_TRUE*) or failure (*JS_FALSE*).

JSBool JS_ReportError()

Description

Describes the reason for a script error. Call this function before returning *JS_FALSE* to give the user information about why the script failed (for example, “wrong number of arguments”).

Arguments

*JSContext *cx*, *char *error*, *size_t sz*

- *cx* is the opaque *JSContext* pointer that was passed to the JavaScript function.
- *error* is a string containing the error message. The string is copied, so the caller should free the string when it is no longer needed. If the string size is not specified (see the *sz* argument, below), then the string must be null-terminated.
- *sz* is the size of the string, in bytes. If *sz* is 0, then the length of the null-terminated string is computed automatically.

Returns

A Boolean value indicating success (*JS_TRUE*) or failure (*JS_FALSE*).

Calling a C function from JavaScript

Once you know how C-level extensibility works in Dreamweaver and the data types and functions it relies on, it's useful to walk through an example of how to build a library and call a function.

This exercise requires three files, which are included in the `Extending/c_files` folder inside the Dreamweaver application folder:

- `mm_jsapi.h` is a header file that includes definitions for the data types and functions described in “The C-level extensibility API” on page 193.
- `Sample.c` is an example file that defines the `computeSum()` function.
- `Sample.mak` is a makefile that you can use to build `Sample.c` into a DLL with Microsoft Visual C++; `Sample.proj` is the equivalent file for building a CFM Library with Metrowerks CodeWarrior. If you are using another tool, you can create the makefile yourself.

To build the DLL in Windows:

- 1 In Microsoft Visual C++, choose `File > Open Workspace` and select `Sample.mak`.
- 2 Choose `Build > Rebuild All`.

When the build operation is complete, a file called `Sample.dll` appears in the folder containing `Sample.mak` (or one of its subfolders).

To build the shared library on the Macintosh:

- 1 Open `Sample.proj` in Metrowerks CodeWarrior.
- 2 Build the project to generate a CFM Library.

When the build operation has finished, a file called `Sample` appears in the folder containing `Sample.proj` (or in one of its subfolders).

To call the `computeSum()` function from the Insert Horizontal Rule object:

- 1 In the Configuration folder within the Dreamweaver application folder, create a folder called JSExtensions.
- 2 Copy `Sample.dll` (Windows) or `Sample` (Macintosh) to the JSExtensions folder.
- 3 In a text editor, open the file called `horizontal_rule.htm` that resides in the Configuration/Objects/Common folder.
- 4 Add the line `alert(Sample.computeSum(2,2));` to the `objectTag()` function so that it appears as follows:

```
function objectTag() {  
    // Return the html tag that should be inserted  
    alert(Sample.computeSum(2,2));  
    return "<HR>";  
}
```

- 5 Save the file and restart Dreamweaver.

To execute the `computeSum()` function:

Choose Insert > Horizontal Rule. A dialog box containing the number 4—the result of computing the sum of 2 plus 2—appears.

CHAPTER 20

The Dreamweaver JavaScript API

The objects, properties, and especially the methods described in “The Document Object Model and JavaScript” on page 13 are a good foundation on which to build extensibility into Dreamweaver. Several tasks unique to authoring environments cannot be accomplished with the methods available in the Netscape, Microsoft, or W3C DOMs, however. The most obvious example of this is selection, which is integral to the user experience in an authoring environment: DOM Level 1 and the proprietary browser DOMs do not address selection because users cannot select and modify content (except in form fields) in a browser window.

To make creating useful Dreamweaver extensions and customizing Dreamweaver menus possible, Dreamweaver exposes more than 400 JavaScript functions to developers beyond the standards-based DOM methods. This represents a huge expansion over what was available in Dreamweaver 3. Almost any task that the user can accomplish in Dreamweaver with the menus, floating panels, inspectors, Site window, or Document window can now be done with JavaScript.

Understanding the objects in the API

All of the custom functions in the following sections are methods of the `dreamweaver` object, the `site` object, or the object that represents a document's DOM. Methods that fit into the latter category are listed here as `dom.functionName()`. To produce the desired results, you must first get the DOM of a document and call the functions as methods of that DOM. You cannot simply type `dom.functionName()`. For example:

```
var currentDOM = dreamweaver.getDocumentDOM('document');
currentDOM.setSelection(100,200);
currentDOM.clipCopy();
var otherDOM =
dreamweaver.openDocument(dreamweaver.getSiteRoot()+
+ "html/foo.htm");
otherDOM.endOfDocument();
otherDOM.clipPaste();
```

Unless otherwise noted, methods of the `dom` object can operate only on open documents, and running a function on a document that isn't open will cause Dreamweaver to report an error. Methods of the `dom` object that can operate only on the active document or on closed documents indicate this fact in their descriptions.

In Dreamweaver 4, `dw` is synonymous with `dreamweaver`. Thus all of the methods of the `dreamweaver` object can be referred to as `dw.functionName()`. This notation appears in code examples throughout this and other chapters.

About enablers

Because every menu item in Dreamweaver is implemented as a JavaScript function, a JavaScript mechanism is required to determine when menu items should be enabled. This mechanism is a series of functions called *enablers*.

An enabler determines whether its associated main function can operate in the current context. For example, `site.canGet()` determines whether Dreamweaver can perform a Get operation (`site.get()`).

Each function specification in this API lists the enabler for the function, if applicable. (Some functions do not have enablers, either because the menu item associated with the function is always enabled, or because the function is unrelated to menus.) For function specifications for the enablers, see “Enablers” on page 465.

How this chapter is organized

The methods in the Dreamweaver JavaScript API are grouped functionally and then alphabetically by object and then by method name. For example, methods that deal with creating, applying, and deleting CSS styles are grouped under CSS Style functions; methods of the `dom` object are listed first, followed by methods of the `dreamweaver` object. Deprecated functions and enablers are listed at the end of this chapter. Optional arguments are enclosed in braces (`{ }`).

Getting document data through the DOM

Virtually all of the functions involved in DOM manipulations require that you first determine which DOM to change. This is the task of `dreamweaver.getDocumentDOM()`, and thus it is the most important function.

`dreamweaver.getDocumentDOM()`

Availability

Dreamweaver 2.0

Description

Provides access to the tree of objects for the specified document. After the tree of objects is returned to the caller, the caller can edit the tree to change the contents of the document.

Arguments

sourceDoc

sourceDoc must be "document", "parent", "parent.frames[number]", "parent.frames['frameName']", or a URL. *document* specifies the document that has the focus and contains the current selection. *parent* specifies the parent frameset (if the currently selected document is in a frame), and *parent.frames[number]* and *parent.frames['frameName']* specify a document that is in a particular frame within the frameset containing the current document. If the argument is a relative URL, it is relative to the extension file. In Dreamweaver 4, *sourceDoc* defaults to *document* if omitted.

Note: If the argument is "document", the caller must be `applyBehavior()`, `deleteBehavior()`, `objectTag()`, or any function in a command or Property Inspector file in order to perform edits to the document.

Returns

The JavaScript document object at the root of the tree.

Enabler

None.

Example

The following code snippet uses `dreamweaver.getDocumentDOM()` to access the background color of the current document:

```
var theDOM = dreamweaver.getDocumentDOM("document");
theDOM.body.bgcolor = "#000000";
```

Assets panel functions

Assets panel functions (programmed into the API as “asset palette”) let you manage and use the elements shown in the Assets panel (these can be templates, libraries, images, Shockwave and Flash movies, URLs, colors, movies, and scripts).

`dreamweaver.assetPalette.addToFavoritesFromDocument()`

Availability

Dreamweaver 4.0

Description

Adds the element selected in the Document window to the Favorites list. This function handles only images, movies, Shockwave files, Flash files, text font colors, and URLs.

Arguments

None.

Returns

Nothing.

`dreamweaver.assetPalette.addToFavoritesFromSiteAssets()`

Availability

Dreamweaver 4.0

Description

Adds elements selected in the Site list to the Favorites list and gives each item a nickname in the Favorites list. This function does not remove the element from the Site list.

Arguments

None.

Returns

Nothing.

`dreamweaver.assetPalette.addToFavoritesFromSiteWindow()`

Availability

Dreamweaver 4.0

Description

Adds the elements selected in the Site window or site map to the Favorites list. This function handles only images, movies, scripts, Shockwave files, Flash files, and URLs (in the case of the site map). If other folders or files are selected, they are ignored.

Arguments

None.

Returns

Nothing.

`dreamweaver.assetPalette.copyToSite()`

Availability

Dreamweaver 4.0

Description

Copies selected elements to another site and puts them in that site's Favorites list. If the elements are files (other than colors or URLs), the actual file is copied into that site.

Arguments

targetSite

targetSite is the name of the target site, as returned from the `site.getSites()` call.

Returns

Nothing.

`dreamweaver.assetPalette.edit()`

Availability

Dreamweaver 4.0

Description

Edits selected elements with primary external editor or Custom Edit control. For colors, the color picker appears. For URLs, a dialog box appears prompting the user for a URL and a nickname. Not available for the site list of colors and URLs.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.assetPalette.canEdit()`

`dreamweaver.assetPalette.getSelectedCategory()`

Availability

Dreamweaver 4.0

Description

Returns the currently selected category, which can be one of the following: "templates", "library", "images", "movies", "shockwave", "flash", "scripts", "colors", or "urls".

Arguments

None.

Returns

The currently selected category.

`dreamweaver.assetPalette.getSelectedItems()`

Availability

Dreamweaver 4.0

Description

Returns an array of the selected items in the Assets panel, either in the site list or Favorites list.

Arguments

None.

Returns

An array of three strings *for each selected item*:

- *name* is the name/file name or nickname as seen in the panel.
- *value* is the full file path, full URL, or color value depending on the item selected.
- *type* is either "folder" or one of the categories: "templates", "library", "images", "movies", "shockwave", "flash", "scripts", "colors", or "urls".

Note: If nothing is selected in the Assets panel, this function returns an array of one empty string.

Example

If "URLs" is the category, and a folder "MyFolderName" and a URL "MyFavoriteURL" are both selected in the Favorites list, the function will return:

```
items[0] = "MyFolderName"
items[1] = " "
items[2] = "folder"
items[3] = "MyFavoriteURL"
items[4] = "http://www.MyFavoriteURL.com"
items[5] = "urls"
```

`dreamweaver.assetPalette.getSelectedView()`

Availability

Dreamweaver 4.0

Description

Indicates which list is currently shown in the Assets panel.

Arguments

None.

Returns

Returns either "site" or "favorites".

dreamweaver.assetPalette.insertOrApply()

Availability

Dreamweaver 4.0

Description

Inserts selected elements or applies the element to the current selection. Applies templates, applies colors to selection, applies URLs to selection, and inserts URLs and other elements at the insertion point. If a document isn't open, the function is not available.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.assetPalette.canEdit()`

dreamweaver.assetPalette.locateInSite()

Availability

Dreamweaver 4.0

Description

Selects files associated with the selected elements in the local side of the Site window. This function does not work for colors or URLs. Available in both the site list and the Favorites list. If a folder is selected in the Favorites list, it will be ignored.

Arguments

None.

Returns

Nothing.

`dreamweaver.assetPalette.newAsset()`

Availability

Dreamweaver 4.0

Description

Creates new element for the current category in the Favorites list. For library and templates, this is a new blank library or template file that the user can name immediately. For colors, the color picker appears. For URLs, a dialog box appears prompting the user for a URL and a nickname. Not available for images, movies, Shockwave files, Flash files, or scripts.

Arguments

None.

Returns

Nothing.

`dreamweaver.assetPalette.newFolder()`

Availability

Dreamweaver 4.0

Description

Creates a new folder in the current category with the default name (untitled) and puts an edit box around the default name. Only available in the Favorites list.

Arguments

None.

Returns

Nothing.

`dreamweaver.assetPalette.recreateLibraryFromDocument()`

Availability

Dreamweaver 4.0

Description

Replaces the deprecated `libraryPalette` function, `recreateLibraryFromDocument()`. Creates an LBI file for the selected instance of a library item in the current document. This function is equivalent to clicking Recreate in the Property inspector.

Arguments

None.

Returns

Nothing.

`dreamweaver.assetPalette.refreshSiteAssets()`

Availability

Dreamweaver 4.0

Description

Scans site, switches to the site list, and populates the list.

Arguments

None.

Returns

Nothing.

`dreamweaver.assetPalette.removeFromFavorites()`

Availability

Dreamweaver 4.0

Description

Removes the selected elements from the Favorites list. This function does not delete the actual file on disk, except in the case of a library or template where the user is prompted before the file is deleted. Only works in the Favorites list or if the category is “Library” or “Templates”.

Arguments

None.

Returns

Nothing.

`dreamweaver.assetPalette.renameNickname()`

Availability

Dreamweaver 4.0

Description

Edits the folder name or the file’s nickname by displaying an edit box around the existing nickname. Only available in the Favorites list or in the Library or Template category.

Arguments

None.

Returns

Nothing.

`dreamweaver.assetPalette.setSelectedCategory()`

Availability

Dreamweaver 4.0

Description

Switches to show a different category.

Arguments

categoryType

categoryType can be one of the following categories: "templates", "library", "images", "movies", "shockwave", "flash", "scripts", "colors", or "urls".

Returns

Nothing.

`dreamweaver.assetPalette.setSelectedView()`

Availability

Dreamweaver 4.0

Description

Switches the display to show either the site list or Favorites list.

Arguments

viewType

viewType can be site or favorites.

Returns

Nothing.

`dreamweaver.referencePalette.getFontSize()`

Availability

Dreamweaver 4.0

Description

Returns the current font size of the Reference panel display region.

Arguments

None.

Returns

The relative font size as small, medium, or large.

`dreamweaver.referencePalette.setFontSize()`

Availability

Dreamweaver 4.0

Description

Changes the font size displayed in the Reference panel.

Arguments

fontSize

fontSize is one of the following relative sizes: `small`, `medium`, or `large`.

Returns

Nothing.

Behavior functions

Behavior functions let you add behaviors to and remove behaviors from an object, find out which behaviors are attached to an object, get information about the object to which a behavior is attached, and so on. Methods of the `dreamweaver.behaviorInspector` object either control or act on the selection in the Behaviors panel, not in the current document.

`dom.addBehavior()`

Availability

Dreamweaver 3.0

Description

Adds a new event/action pair to the selected element. This function is valid only for the active document.

Arguments

event, *action*, [*eventBasedIndex*]

- *event* is the JavaScript event handler that should be used to attach the behavior to the element; for example, `onClick`, `onMouseOver`, or `onLoad`.
- *action* is the function call that would be returned by `applyBehavior()` if the action were added using the Behaviors panel, for example, `"MM_popupMsg('Hello World')"`.
- *eventBasedIndex* is the position at which this action should be added. *eventBasedIndex* is a zero-based index; if two actions already are associated with the specified event, and you specify *eventBasedIndex* as 1, this action will be executed between the other two. If this argument is omitted, the action is added after all existing actions for the specified event.

Returns

Nothing.

Enabler

None.

dom.getBehavior()**Availability**

Dreamweaver 3.0

Description

Gets the action at the specified position within the specified event. This function acts on the current selection and is valid only for the active document.

Arguments

event, {*eventBasedIndex*}

- *event* is the JavaScript event handler through which the action is attached to the element; for example, `onClick`, `onMouseOver`, or `onLoad`.
- *eventBasedIndex* is the position of the action to get. For example, if two actions are associated with the specified event, 0 is the first one and 1 is the second. If this argument is omitted, all the actions for the specified event are returned.

Returns

A string representing the function call (for example, `"MM_swapImage('document.Image1','document.Image1','foo.gif','#933292969950')"`), or an array of strings if *eventBasedIndex* was omitted.

Enabler

None.

dom.reapplyBehaviors()

Availability

Dreamweaver 3.0

Description

Checks to make sure that the functions associated with any behavior calls on the specified node are in the HEAD of the document, and inserts them if they are missing.

Arguments

{elementNode}

elementNode is an element node within the current document. If the argument is omitted, Dreamweaver checks all element nodes in the document for orphaned behavior calls.

Returns

Nothing.

Enabler

None.

dom.removeBehavior()

Availability

Dreamweaver 3.0

Description

Removes the action at the specified position within the specified event. This function acts on the current selection and is valid only for the active document.

Arguments

event, {eventBasedIndex}

- *event* is the event handler through which the action is attached to the element; for example, `onClick`, `onMouseOver`, or `onLoad`. If this argument is omitted, all actions are removed from the element.
- *eventBasedIndex* is the position of the action to be removed. For example, if two actions are associated with the specified event, 0 is the first one and 1 is the second. If this argument is omitted, all the actions for the specified event are removed.

Returns

Nothing.

Enabler

None.

`dreamweaver.getBehaviorElement()`

Availability

Dreamweaver 2.0

Description

Gets the DOM object corresponding to the tag to which the behavior is being applied. This function is applicable only in Behavior action files.

Arguments

None.

Returns

A DOM object or `null`. This function returns `null` under the following circumstances:

- When the current script is not executing within the context of the Behaviors panel.
- When the Behaviors panel is being used to edit a behavior in a timeline.
- When the currently executing script was invoked by `dreamweaver.popupAction()`.
- When the Behaviors panel is attaching an event to a link wrapper and the link wrapper does not yet exist.
- When this function appears outside of an action file.

Enabler

None.

Example

`dreamweaver.getBehaviorElement()` can be used in the same way as `dreamweaver.getBehaviorTag()` to determine whether the selected action is appropriate for the selected HTML tag, except that it gives you access to more information about the tag and its attributes. For example, if you write an action that can be applied only to a hyperlink (A HREF) that does not target another frame or window, you can use `getBehaviorElement()` as part of the function that initializes the user interface for the Parameters dialog box:

```
function initializeUI(){
    var theTag = dreamweaver.getBehaviorElement();
    var CANBEAPPLIED = (theTag.tagName == "A" && ~
theTag.getAttribute("HREF") != null && ~
theTag.getAttribute("TARGET") == null);
    if (CANBEAPPLIED) {
        // display the action UI
    } else{
        // display a helpful message that tells the user
        // that this action can only be applied to a
        // hyperlink without an explicit target]
    }
}
```

`dreamweaver.getBehaviorTag()`

Availability

Dreamweaver1.2

Description

Gets the source of the tag to which the behavior is being applied. This function is applicable only in action files.

Arguments

None.

Returns

A string representing the source of the tag. This is the same string that is passed as an argument (*HTMLelement*) to the `canAcceptBehavior()` function. If this function appears outside of an action file, the return value is an empty string.

Enabler

None.

Example

If you write an action that can be applied only to a hyperlink (A HREF), you can use `getBehaviorTag()` as part of the function that initializes the user interface for the Parameters dialog box:

```
function initializeUI(){
    var theTag = dreamweaver.getBehaviorTag().toUpperCase();
    var CANBEAPPLIED = (theTag.indexOf('HREF') != -1);
    if (CANBEAPPLIED) {
        // display the action UI
    } else{
        // display a helpful message that tells the user
        // that this action can only be applied to a
        // hyperlink
    }
}
```


`dreamweaver.popupAction()`

Availability

Dreamweaver 2.0

Description

Presents the user with a Parameters dialog box for the specified behavior action. To the user, the effect is the same as selecting the action from the Actions pop-up menu in the Behaviors panel. This function lets extension files other than actions attach behaviors to objects in the user's document. It blocks other edits until the user dismisses the dialog box.

Note: This function can be called only within `objectTag()` or in any script in a command or Property Inspector file.

Arguments

actionName, {*funcCall*}

- *actionName* is the name of a file in the Configuration/Behaviors/Actions folder that contains a JavaScript behavior action (for example, "Timeline/PlayTimeline.htm").
- *funcCall* is a string containing a function call for the action specified in *actionName* (for example, "MM_playTimeline(...)"). This argument, if specified, is supplied by the `applyBehavior()` function in the action file.

Returns

The function call for the behavior action. When the user clicks OK in the Parameters dialog box, the behavior is added to the current document (the appropriate functions are added to the HEAD of the document, HTML may be added to the top of the BODY, and other edits may be made to the document). The function call (for example, "MM_playTimeline(...)") is not added to document, but becomes the return value of this function.

Enabler

None.

`dreamweaver.behaviorInspector.getBehaviorAt()`

Availability

Dreamweaver 3.0

Description

Gets the event/action pair at the specified position in the Behaviors panel.

Arguments

positionIndex

Returns

An array of two items:

- An event handler
- A function call or JavaScript statement

Enabler

None.

Example

Because *positionIndex* is a zero-based index, if the Behaviors panel displays the list shown, a call to `dw.behaviorInspector.getBehaviorAt(2)` returns an array containing two strings: "onMouseOver" and "MM_changeProp('document.moon','document.moon','src','sun.gif','IMG')".

`dreamweaver.behaviorInspector.getBehaviorCount()`

Availability

Dreamweaver 3.0

Description

Counts the number of actions attached to the currently selected element through event handlers.

Arguments

None.

Returns

An integer representing the number of actions attached to the element. This number is equivalent to the number of actions visible in the Behaviors panel and includes both Dreamweaver behavior actions and custom JavaScript.

Enabler

None.

Example

A call to `dw.behaviorInspector.getBehaviorCount()` for the selected link `` would return 2.

`dreamweaver.behaviorInspector.getSelectedBehavior()`

Availability

Dreamweaver 3.0

Description

Gets the position in the Behaviors panel of the selected action.

Arguments

None.

Returns

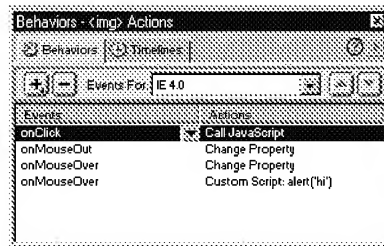
An integer representing the position in the Behaviors panel of the selected action, or -1 if no action is selected.

Enabler

None.

Example

If the first action in the Behaviors panel is selected, as shown, a call to `dw.behaviorInspector.getSelectedBehavior()` returns 0.



`dreamweaver.behaviorInspector.moveBehaviorDown()`

Availability

Dreamweaver 3.0

Description

Moves a behavior action lower in sequence by changing its execution order within the scope of an event.

Arguments

positionIndex

positionIndex is the position of the action in the Behaviors panel. The first action in the list is at position 0.

Returns

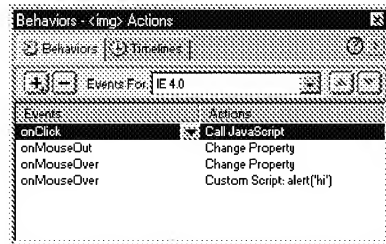
Nothing.

Enabler

None.

Example

Assuming the Behaviors panel setup shown in this example, calling `dw.behaviorInspector.moveBehaviorDown(2)` would swap the positions of the Custom Script and the Change Property actions on the `onMouseOver` event. Calling `dw.behaviorInspector.moveBehaviorDown()` for any other position would have no effect because the `onClick` and `onMouseOut` events each have only one behavior associated with them, and the behavior at position 3 is already at the bottom of the `onMouseOver` group.



dreamweaver.behaviorInspector.moveBehaviorUp()

Availability

Dreamweaver 3.0

Description

Moves a behavior higher in sequence by changing its execution order within the scope of an event.

Arguments

positionIndex

positionIndex is the position of the action in the Behaviors panel. The first action in the list is at position 0.

Returns

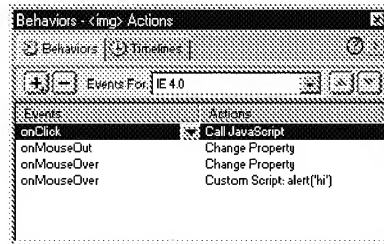
Nothing.

Enabler

None.

Example

Assuming the Behaviors panel setup shown in this example, calling `dw.behaviorInspector.moveBehaviorUp(3)` would swap the positions of the Custom Script and the Change Property actions on the `onMouseOver` event. Calling `dw.behaviorInspector.moveBehaviorUp()` for any other position would have no effect because the `onClick` and `onMouseOut` events each have only one behavior associated with them, and the behavior at position 2 is already at the top of the `onMouseOver` group.



`dreamweaver.behaviorInspector.setSelectedBehavior()`

Availability

Dreamweaver 3.0

Description

Selects the action at the specified position in the Behaviors panel.

Arguments

positionIndex

positionIndex is the position of the action in the Behaviors panel. The first action in the list is at position 0. To deselect all actions, specify a *positionIndex* of -1. Specifying a position for which no action exists is equivalent to specifying -1.

Returns

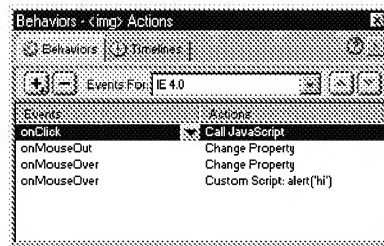
Nothing.

Enabler

None.

Example

Assuming the Behaviors panel setup shown in this example, calling `dw.behaviorInspector.setSelection(3)` would select the Change Property action associated with the `onMouseOut` event.



Clipboard functions

Clipboard functions are all related to cutting, copying, and pasting. On the Macintosh, some clipboard functions can also apply to edit fields in dialog boxes and floating panels. Functions that can operate in edit fields are implemented both as methods of the `dreamweaver` object and as methods of the `dom` object. The `dreamweaver` version of the function operates on the selection in the window that has focus: the current Document window, the Code inspector, or the Site window. On the Macintosh, the function can also operate on the selection in an edit field if the field has focus. The `dom` version of the function always operates on the selection in the specified document.

`dom.clipCopy()`

Availability

Dreamweaver 3.0

Description

Copies the selection, including any HTML markup that defines the selection, to the clipboard.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.clipCopyText()

Availability

Dreamweaver 3.0

Description

Copies the selected text to the clipboard, ignoring any HTML markup.

Arguments

None.

Returns

Nothing.

Enabler

dom.canClipCopyText()

dom.clipCut()

Availability

Dreamweaver 3.0

Description

Removes the selection, including any HTML markup that defines the selection, to the clipboard.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.clipPaste()

Availability

Dreamweaver 3.0

Description

Pastes the contents of the clipboard into the current document at the current insertion point or in place of the current selection. If the clipboard contains HTML, it is interpreted as such.

Arguments

None.

Returns

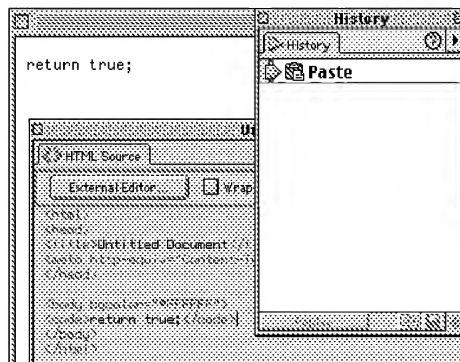
Nothing.

Enabler

dom.canClipPaste()

Example

If the clipboard contains `<code>return true;</code>`, a call to `dw.getDocumentDOM().clipPaste()` would result in the following:



dom.clipPasteText()

Availability

Dreamweaver 3.0

Description

Pastes the contents of the clipboard into the current document at the current insertion point or in place of the current selection, replacing any linefeeds in the clipboard content with BR tags. If the clipboard contains HTML, it is not interpreted; angle brackets are pasted as < and >.

Arguments

None.

Returns

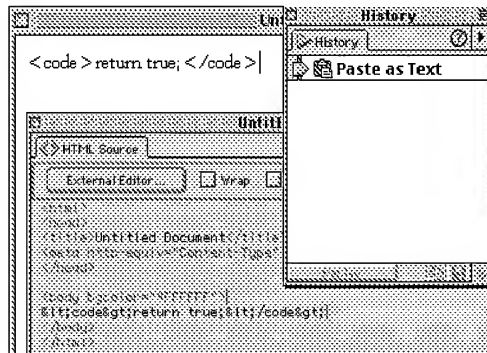
Nothing.

Enabler

dom.canClipPasteText()

Example

If the clipboard contains `<code>return true;</code>`, a call to `dw.getDocumentDOM().clipPasteText()` would result in the following:



`dreamweaver.clipCopy()`

Availability

Dreamweaver 3.0

Description

Copies the current selection (from whichever Document window, dialog box, floating panel, or Site window pane has focus) to the clipboard.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.canClipCopy()`

`dreamweaver.clipCut()`

Availability

Dreamweaver 3.0

Description

Removes the selection (from whichever Document window, dialog box, floating panel, or Site window pane has focus) to the clipboard.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.canClipCut()`

dreamweaver.clipPaste()

Availability

Dreamweaver 3.0

Description

Pastes the contents of the clipboard into the current document, dialog box, floating panel, or Site window pane.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.canClipPaste()`

dreamweaver.getClipboardText()

Availability

Dreamweaver 3.0

Description

Gets all the text that is stored on the clipboard.

Arguments

[bAsText]

[bAsText] is a Boolean value that specifies whether or not the clipboard content is retrieved as text. If *bAsText* is true, the clipboard content is retrieved as text. If *bAsText* is false, the behavior is the same as in Dreamweaver 3. This argument defaults to false.

Returns

A string containing the contents of the clipboard, if the clipboard contains text (which may be HTML); otherwise, nothing.

Enabler

None.

Example

If `dw.getClipboardText()` returns "text bold text", then `dw.getClipboardText(true)` returns "text bold text".

Command functions

Command functions help you make the most of the files in the Configuration/Commands folder. They manage the Command menu and call commands from other types of extension files.

`dreamweaver.editCommandList()`

Availability

Dreamweaver 3.0

Description

Opens the Edit Command List dialog box.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.runCommand()`

Availability

Dreamweaver 3.0

Description

Executes the specified command. To the user, the effect is the same as choosing the command from a menu; if a dialog box is associated with the command, it appears (and the command script blocks other edits until the user dismisses the dialog box). This function provides the ability to call a command from another extension file.

Note: This function can be called only within the `objectTag()` function or in any script in a command, Property Inspector file, or menu.

Arguments

commandFile, {*commandArg1*}, {*commandArg2*},...{*commandArgN*}

- *commandFile* is the name of a file in the Configuration/Commands folder.
- The second and remaining arguments are passed to *commandFile* as arguments.

Returns

Nothing.

Enabler

None.

Example

You can write a custom Property inspector for tables that lets users get to the Format Table command from a button on the inspector by calling the following function from the button's `onClick` event handler:

```
function callFormatTable(){  
    dw.runCommand('Format Table.htm');  
}
```

Conversion functions

Conversion functions deal with converting tables to layers, layers to tables, and Cascading Style Sheets (CSS) styles to HTML markup. Each function exactly duplicates the behavior of one of the conversion commands in the File or Modify menu.

`dom.convertLayersToTable()`**Availability**

Dreamweaver 3.0

Description

Opens the Convert Layers to Table dialog box.

Arguments

None.

Returns

Nothing.

Enabler

`dom.canConvertLayersToTable()`

dom.convertTablesToLayers()

Availability

Dreamweaver 3.0

Description

Opens the Convert Tables to Layers dialog box.

Arguments

None.

Returns

Nothing.

Enabler

dom.canConvertTablesToLayers()

dom.convertTo30()

Availability

Dreamweaver 3.0

Description

Opens the Convert to 3.0 Compatible dialog box.

Arguments

None.

Returns

Nothing.

Enabler

None.

CSS style functions

CSS style functions handle the application, removal, creation, and deletion of CSS styles. Methods of the `dreamweaver.cssStylePalette` object either control or act on the selection in the Style panel, not in the current document.

`dom.applyCSSStyle()`

Availability

Dreamweaver 4.0

Description

Applies the specified style to the specified element. This function is valid only for the active document.

Arguments

elementNode, *styleName*, [*classOrID*], [*bForceNesting*]

- *elementNode* is an element node in the DOM. If *elementNode* is specified as NULL or empty string (''), the function acts on the current selection.
- *styleName* is the name of a CSS style.
- [*classOrID*] is the attribute with which the style should be applied (either “class” or “id”). If *elementNode* is specified as NULL or empty string and no tag exactly surrounds the selection, the style is applied using SPAN tags. If the selection is an insertion point, Dreamweaver uses heuristics to determine to which tag the style should be applied.
- [*bForceNesting*] is a Boolean value, which indicates whether or not nesting is allowed. If the *bForceNesting* flag is specified, Dreamweaver inserts a new SPAN tag instead of trying to modify the existing tags in the document. This argument defaults to `false` if not specified.

Returns

Nothing.

Enabler

None.

Example

The following code applies the red style to the selection, either by surrounding the selection with SPAN tags or by applying a CLASS attribute to the tag that surrounds the selection:

```
var theDOM = dreamweaver.getDocumentDOM('document');
theDOM.applyCSSStyle('', 'red');
```


`dom.removeCSSStyle()`

Availability

Dreamweaver 3.0

Description

Removes the CLASS or ID attribute from the specified element, or removes the SPAN tag that completely surrounds the specified element. This function is valid only for the active document.

Arguments

elementNode, {*classOrID*}

- *elementNode* is an element node in the DOM. If *elementNode* is specified as an empty string (''), the function acts on the current selection.
- *classOrID* is the attribute that should be removed (either "class" or "id"). If *classOrID* is not specified, it defaults to "class". If no CLASS attribute is defined for *elementNode*, then the SPAN tag surrounding *elementNode* is removed.

Returns

Nothing.

Enabler

None.

`dreamweaver.stylePalette.attachExternalStylesheet()`

Availability

Dreamweaver 4.0

Description

Displays a dialog box allowing users to attach an external style sheet.

Arguments

None.

Returns

Nothing.

`dreamweaver.cssStylePalette.deleteSelectedStyle()`

Availability

Dreamweaver 3.0

Description

Deletes from the document the style that is currently selected in the Style panel.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.cssStylePalette.duplicateSelectedStyle()`

Availability

Dreamweaver 3.0

Description

Duplicates the style that is currently selected in the Style panel and displays the Duplicate Style dialog box to let the user assign a name or selector to the new style.

Arguments

None.

Returns

Nothing.

Enabler

None.

dreamweaver.cssStylePalette.editSelectedStyle()

Availability

Dreamweaver 3.0

Description

Opens the Style Definition dialog box for the style that is currently selected in the Style panel.

Arguments

None.

Returns

Nothing.

Enabler

None.

dreamweaver.cssStylePalette.editStyleSheet()

Availability

Dreamweaver 3.0

Description

Opens the Edit Style Sheet dialog box.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.cssStylePalette.getSelectedStyle()`

Availability

Dreamweaver 3.0

Description

Gets the name of the style that is currently selected in the Style panel.

Arguments

None.

Returns

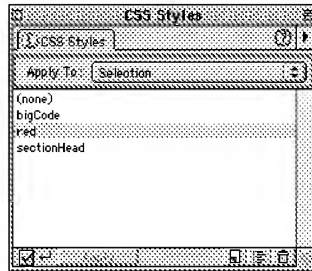
A string representing the name of the style, or an empty string if no style is selected.

Enabler

None.

Example

If the style `red` is selected, as shown, a call to `dw.cssStylePalette.getSelectedStyle()` returns `"red"`.



`dreamweaver.cssStylePalette.getSelectedTarget()`

Availability

Dreamweaver 3.0

Description

Gets the selected element in the Apply To pop-up menu at the top of the Style panel.

Arguments

None.

Returns

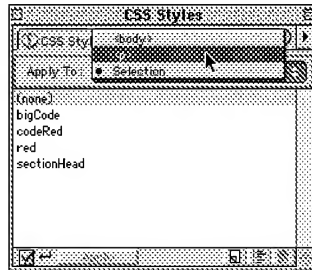
The object to which the style should be applied, or NULL if the target is the current selection.

Enabler

None.

Example

Before applying a style, use `dw.cssStylePalette.getSelectedTarget()` in case the user has changed the target, as shown.



For example:

```
var currDOM = dw.getDocumentDOM();
currDOM.applyCSSStyle(dw.cssStylePalette.getSelectedTarget(), "codeRed");
```

`dreamweaver.cssStylePalette.getStyles()`

Availability

Dreamweaver 3.0

Description

Gets a list of all the class styles in the active document.

Arguments

None.

Returns

An array of strings representing the names of all class styles in the document.

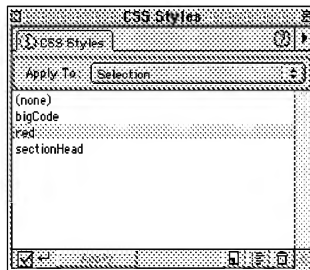
Enabler

None.

Example

Assuming the CSS Styles panel setup shown in this example, a call to `dw.cssStylePalette.getStyles()` would return an array containing the following strings:

- "bigCode"
- "red"
- "sectionHead"



`dreamweaver.cssStylePalette.newStyle()`

Availability

Dreamweaver 3.0

Description

Opens the New Style dialog box.

Arguments

None.

Returns

Nothing.

Enabler

None.

External application functions

External application functions handle operations related to the browsers and external editors defined in the Preview in Browser and External Editors preferences. These functions let you get information about these external applications and open files with them.

`dreamweaver.browseDocument()`

Availability

Dreamweaver 2.0, enhanced in 3.0 and 4.0.

Description

Opens the specified URL in the specified browser.

Arguments

fileName, {*browser*}

- *fileName* is the name of the file to be opened, expressed as an absolute URL.
- *browser*, added in Dreamweaver 3, specifies which browser to use. This argument can be the name of a browser as defined in the Preview in Browser preferences or either 'primary' or 'secondary'. If omitted, the URL is opened in the user's primary browser.

Returns

Nothing.

Enabler

None.

Example

The following function uses `dreamweaver.browseDocument()` to open the Hotwired home page in a browser:

```
function goToHotwired(){
    dreamweaver.browseDocument('http://www.hotwired.com/');
}
```

In Dreamweaver 4, you can expand this operation to open the document in Internet Explorer using the following code:

```
function goToHotwired(){
    var prevBrowsers = dw.getBrowserList();
    var theBrowser = "";
    for (var i=1; i < prevBrowsers.length; i+2){
        if (prevBrowsers[i].indexOf('Iexplore.exe') != -1){
            theBrowser = prevBrowsers[i];
            break;
        }
    }
    dw.browseDocument('http://www.hotwired.com/',theBrowser);
}
```

For more information on `dw.getBrowserList()`, see “`dreamweaver.getBrowserList()`” on page 244.

dreamweaver.getBrowserList()**Availability**

Dreamweaver 3.0

Description

Gets a list of all the browsers in the Preview in Browser submenu.

Arguments

None.

Returns

An array containing a pair of strings for each browser in the list. The first string in each pair is the name of the browser, and the second string is its location on the user's machine, expressed as a file:// URL. If no browsers appear in the submenu, the function returns nothing.

Enabler

None.

`dreamweaver.getExtensionEditorList()`

Availability

Dreamweaver 3.0

Description

Gets a list of editors for the specified file from the External Editors preferences.

Arguments

fileURL

fileURL can be a complete file:// URL, a file name, or a file extension (including the period).

Returns

An array containing a pair of strings for each editor in the list. The first string in each pair is the name of the editor, and the second string is its location on the user's machine, expressed as a file:// URL. If no editors appear in the preferences, the function returns an array of one empty string.

Enabler

None.

Example

A call to `dw.getExtensionEditorList(".gif")` might return an array containing the following strings:

- "Fireworks 3"
- "file:///C:/Program Files/Macromedia/Fireworks 3/Fireworks 3.exe"

`dreamweaver.getExternalTextEditor()`

Availability

Dreamweaver 4.0

Description

Gets the name of the currently configured external text editor.

Arguments

None.

Returns

A string containing the name of the text editor suitable for presentation in the UI, not the full path.

Enabler

None.

`dreamweaver.getPrimaryBrowser()`

Availability

Dreamweaver 3.0

Description

Gets the path to the primary browser.

Arguments

None.

Returns

A string containing the path on the user's hard drive to the primary browser, expressed as a file:// URL, or nothing if no primary browser is defined.

Enabler

None.

`dreamweaver.getPrimaryExtensionEditor()`

Availability

Dreamweaver 3.0

Description

Gets the primary editor for the specified file.

Arguments

fileURL

Returns

An array containing a pair of strings. The first string in the pair is the name of the editor, and the second string is its location on the user's machine, expressed as a file:// URL. If no primary editor is defined, the function returns an array of one empty string.

Enabler

None.

`dreamweaver.getSecondaryBrowser()`

Availability

Dreamweaver 3.0

Description

Gets the path to the secondary browser.

Arguments

None.

Returns

A string containing the path on the user's hard disk to the secondary browser, expressed as a file:// URL, or nothing if no secondary browser is defined.

Enabler

None.

`dreamweaver.openWithApp()`

Availability

Dreamweaver 3.0

Description

Opens the specified file with the specified application.

Arguments

fileURL, *appURL*

- *fileURL* is the path to the file to be opened, expressed as a file:// URL.
- *appURL* is the path to the application that is to open the file, expressed as a file:// URL.

Returns

Nothing.

Enabler

None.

`dreamweaver.openWithBrowseDialog()`

Availability

Dreamweaver 3.0

Description

Opens the Select External Editor dialog box to let the user choose the application with which to open the specified file.

Arguments

fileURL

Returns

Nothing.

Enabler

None.

`dreamweaver.openWithExternalTextEditor()`

Availability

Dreamweaver 3.0

Description

Opens the current document in the external text editor specified in the External Editors preferences.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.openWithImageEditor()`

Availability

Dreamweaver 3.0

Description

Opens the specified file with the specified image editor.

Note: This function invokes a special Fireworks integration mechanism that returns information to the active document if Fireworks is specified as the image editor. To prevent errors if no document is active, this function should never be called from the Site window.

Arguments

fileURL, *appURL*

- *fileURL* is the path to the file to be opened, expressed as a file:// URL.
- *appURL* is the path to the application with which to open the file, expressed as a file:// URL.

Returns

Nothing.

Enabler

None.

File manipulation functions

File manipulation functions deal with creating, opening, and saving documents, and with exporting cascading style sheets to external files. They accomplish such tasks as browsing for files or folders, creating files based on templates, closing documents, and getting information about recently opened files.

`dreamweaver.browseForFileURL()`

Availability

Dreamweaver 1.0, enhanced in 2.0, 3.0 and 4.0

Description

Opens the specified type of dialog box with the specified label in the title bar.

Arguments

openSelectOrSave, {*titleBarLabel*}, {*bShowPreviewPane*},
{*bSuppressSiteRootWarnings*}, {*arrayOfExtensions*}

- *openSelectOrSave* indicates the type of dialog box: open, select, or save.
- *titleBarLabel*, added in Dreamweaver 2, is the label that should appear in the title bar of the dialog box. If this argument is omitted, Dreamweaver uses the default label supplied by the operating system.
- *bShowPreviewPane*, added in Dreamweaver 2, is a Boolean value indicating whether to display the image preview pane in the dialog box. If this argument is true, the dialog box filters for image files; if omitted, it defaults to false.
- *bSuppressSiteRootWarnings*, added in Dreamweaver 3, is a Boolean value indicating whether to suppress warnings about the selected file being outside the site root. If this argument is omitted, it defaults to false.
- *arrayOfExtensions*, added in Dreamweaver 4, is an array of strings for specifying the “Files of type” list menu default appearance at the bottom of the dialog box. The proper syntax is `menuEntryText|.xxx[;.yyy;.zzz]|CCCC|`, where *menuEntryText* is the name of the file type to appear. The extensions can be specified as `.xxx[;.yyy;.zzz]` or `CCCC`, where `.xxx` specifies the file extension for the file type (optionally, `.yyy` and `.zzz` specify multiple file extensions) and `CCCC` is the four-character file type constant for use on the Macintosh.

Returns

A string containing the name of the file, expressed as a file:// URL.

Enabler

None.

dreamweaver.browseForFolderURL()

Availability

Dreamweaver 3.0

Description

Opens the Choose Folder dialog box with the specified label in the title bar.

Arguments

{titleBarLabel}, {directoryToStartIn}

- *titleBarLabel* is the label that should appear in the title bar of the dialog box. If omitted, *titleBarLabel* defaults to “Choose Folder.”
- *directoryToStartIn* is the path to the directory to start in, expressed as a file:// URL.

Returns

A string containing the name of the folder, expressed as a file:// URL.

Enabler

None.

Example

The following code returns the URL of a folder:

```
return dw.browseForFolderURL('Select a Folder',  
dw.getSiteRoot());
```

dreamweaver.closeDocument()

Availability

Dreamweaver 3.0

Description

Closes the specified document.

Arguments

documentObject

documentObject is the object at the root of a document's DOM tree (the value returned by `dreamweaver.getDocumentDOM()`). If *documentObject* refers to the active document, the Document window might not close until the script that calls this function finishes executing.

Returns

Nothing.

Enabler

None.

dreamweaver.createDocument()

Availability

Dreamweaver 2.0

Description

Opens a new document either in the same window or in a new window. The new document becomes the active document.

Note: This function can be called only from menus.xml or a command or Property Inspector file. If a behavior action or object tries to call this function, Dreamweaver displays an error message.

Arguments

{bOpenInSameWindow}

bOpenInSameWindow is a Boolean value indicating whether to open the new document in the current window. If *bOpenInSameWindow* is *false* or omitted, or the function is called on the Macintosh, the new document opens in a separate window.

Returns

The document object for the newly created document. This is the same value returned by `dreamweaver.getDocumentDOM()`.

Enabler

None.

dreamweaver.exportCSS()

Availability

Dreamweaver 3.0

Description

Opens the Export Styles as CSS File dialog box.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.canExportCSS()`

`dreamweaver.exportEditableRegionsAsXML()`

Availability

Dreamweaver 3.0

Description

Opens the Export Editable Regions as XML dialog box.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.getRecentFileList()`

Availability

Dreamweaver 3.0

Description

Gets a list of all the files in the recent files list at the bottom of the File menu.

Arguments

None.

Returns

An array of strings representing the paths of the most recently accessed files, expressed as file:// URLs. If there are no recent files, the function returns nothing.

Enabler

None.

`dreamweaver.importXMLIntoTemplate()`

Availability

Dreamweaver 3.0

Description

Opens the Import XML dialog box.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.newFromTemplate()`

Availability

Dreamweaver 3.0

Description

Creates a new document from the specified template. If no argument is supplied, the Select Template dialog box appears.

Arguments

{templateURL}, bmaintain

- *templateURL* is the path to a template in the current site, expressed as a file: // URL.
- *bmaintain* is a Boolean, true or false, indicating whether or not to maintain the link to the original template.

Returns

Nothing.

Enabler

None.

`dreamweaver.openDocument()`

Availability

Dreamweaver 2.0

Description

Opens a document for editing in a new Dreamweaver window and gives it the focus. To the user, the effect is the same as choosing File > Open and selecting a file. If the specified file is already open, the window containing the document comes to the front. The window containing the specified file becomes the currently selected document. In Dreamweaver 2, if check in/check out is enabled, the file is checked out before it is opened. In Dreamweaver 4, you must use `dreamweaver.openDocumentFromSite()` to get this behavior.

Note: This function cannot be called from Behavior action or object files. An error will result.

Arguments

fileName

fileName is the name of the file to be opened, expressed as a URL. If the URL is relative, it is relative to the file containing the script that called this function.

Returns

The document object for the specified file. This is the same value that is returned by `dreamweaver.getDocumentDOM()`.

Enabler

None.

`dreamweaver.openDocumentFromSite()`**Availability**

Dreamweaver 3.0

Description

Opens a document for editing in a new Dreamweaver window and gives it the focus. To the user, the effect is the same as double-clicking a file in the Site window. If the specified file is already open, the window containing the document comes to the front. The window containing the specified file becomes the currently selected document.

Note: This function cannot be called from Behavior action or object files. An error will result.

Arguments

fileName

fileName is the name of the file to be opened, expressed as a URL. If the URL is relative, it is relative to the file containing the script that called this function.

Returns

The document object for the specified file. This is the same value that is returned by `dreamweaver.getDocumentDOM()`.

Enabler

None.

`dreamweaver.openInFrame()`**Availability**

Dreamweaver 3.0

Description

Opens the Open In Frame dialog box. When the user selects a document, it is opened into the active frame.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.canOpenInFrame()`

`dreamweaver.releaseDocument()`

Availability

Dreamweaver 2.0

Description

Explicitly releases a previously referenced document from memory.

Documents referenced by `dreamweaver.getObjectTags()`, `dreamweaver.getObjectRefs()`, `dreamweaver.getDocumentPath()`, or `dreamweaver.getDocumentDOM()` are automatically released when the script that contains the call finishes executing. If the script opens many documents, you must use this function to explicitly release documents before finishing the script to avoid running out of memory.

Note: This function is relevant only for documents that were referenced by a URL, that are not currently open in a frame or Document window, and that are not extension files. (Extension files are loaded into memory at startup and are not released until you quit Dreamweaver.)

Arguments

documentObject

documentObject is the object at the root of a document's DOM tree (the value returned by `dreamweaver.getDocumentDOM()`).

Returns

Nothing.

Enabler

None.

`dreamweaver.revertDocument()`

Availability

Dreamweaver 3.0

Description

Reverts the specified document to the previously saved version.

Arguments

documentObject

documentObject is the object at the root of a document's DOM tree (the value returned by `dreamweaver.getDocumentDOM()`).

Returns

Nothing.

Enabler

`dreamweaver.canRevertDocument()`

`dreamweaver.saveAll()`

Availability

Dreamweaver 3.0

Description

Saves all open documents, opening the Save As dialog box for any documents that have not previously been saved.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.canSaveAll()`

`dreamweaver.saveDocument()`

Availability

Dreamweaver 2.0

Description

Saves the specified file on a local drive.

Note: In Dreamweaver 2, if the file is read-only, Dreamweaver tries to check it out. If the document is still read-only after this attempt—or if it cannot be created—an error message appears.

Arguments

documentObject, {*fileURL*}

- *documentObject* is the object at the root of a document's DOM tree (the value returned by `dreamweaver.getDocumentDOM()`).
- *fileURL* is a URL representing a location on a local drive. If the URL is relative, it is relative to the extension file. In Dreamweaver 2, this argument is required. If *fileURL* is omitted in Dreamweaver 4, the file is saved to its current location if it has been previously saved; otherwise, a Save dialog box appears.

Returns

A Boolean value indicating success (`true`) or failure (`false`).

Enabler

`dreamweaver.canSaveDocument()`

dreamweaver.saveDocumentAs()

Availability

Dreamweaver 3.0

Description

Opens the Save As dialog box.

Arguments

documentObject

documentObject is the object at the root of a document's DOM tree (the value returned by `dreamweaver.getDocumentDOM()`).

Returns

Nothing.

Enabler

None.

dreamweaver.saveDocumentAsTemplate()

Availability

Dreamweaver 3.0

Description

Opens the Save As Template dialog box.

Arguments

documentObject

documentObject is the object at the root of a document's DOM tree (the value returned by `dreamweaver.getDocumentDOM()`).

Returns

Nothing.

Enabler

`dreamweaver.canSaveDocumentAsTemplate()`

dreamweaver.saveFrameset()

Availability

Dreamweaver 3.0

Description

Saves the specified frameset, or opens the Save As dialog box if the frameset has not previously been saved.

Arguments

documentObject

documentObject is the object at the root of a document's DOM tree (the value returned by `dreamweaver.getDocumentDOM()`).

Returns

Nothing.

Enabler

`dreamweaver.canSaveFrameset()`

dreamweaver.saveFramesetAs()

Availability

Dreamweaver 3.0

Description

Opens the Save As dialog box for the frameset file that includes the specified DOM.

Arguments

documentObject

documentObject is the object at the root of a document's DOM tree (the value returned by `dreamweaver.getDocumentDOM()`).

Returns

Nothing.

Enabler

`dreamweaver.canSaveFramesetAs()`

Find/replace functions

Find/replace functions handle find and replace operations. They cover both basic functionality, such as finding the next instance of a search pattern, and complex replacement operations that require no user interaction.

`dreamweaver.findNext()`

Availability

Dreamweaver 3.0

Description

Finds the next instance of the search string that was specified previously by `dreamweaver.setUpFind()`, by `dreamweaver.setUpComplexFind()`, or by the user in the Find dialog box, and selects the instance in the document.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.canFindNext()`

`dreamweaver.replace()`

Availability

Dreamweaver 3.0

Description

Verifies that the current selection matches the search criteria that was specified previously by `dreamweaver.setUpFindReplace()`, by `dreamweaver.setUpComplexFindReplace()`, or by the user in the Replace dialog box; then replaces it with the content specified in that query.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.replaceAll()`

Availability

Dreamweaver 3.0

Description

Replaces each section of the current document that matches the search criteria that was specified previously by `dreamweaver.setUpFindReplace()`, by `dreamweaver.setUpComplexFindReplace()`, or by the user in the Replace dialog box, with the content specified in that query.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.setUpComplexFind()`

Availability

Dreamweaver 3.0

Description

Prepares for an advanced text or tag search by loading the specified XML query.

Arguments

xmlQueryString

xmlQueryString is a string of XML code beginning with `<dwquery>` and ending with `</dwquery>`. (To get a string of the proper format, set up the query in the Find dialog box, click the Save Query button, open the query file in a text editor, and copy everything from the beginning of the `<dwquery>` tag to the end of the `</dwquery>` tag.)

Returns

Nothing.

Enabler

None.

Example

The first line of the following code sets up a tag search and specifies that the scope of the search should be the current document; the second line performs the search operation:

```
dw.setUpComplexFind('<dwquery><queryparams matchcase="false" ↵
ignorewhitespace="true" useregexp="false"/><find><qtag qname="a">↵
<qattribute qname="href" qcompare="=" qvalue="#"></qattribute>↵
<qattribute qname="onMouseOut" qcompare="=" qvalue="" qnegate="true">↵
</qattribute></qtag></find></dwquery>'); dw.findNext();
```

dreamweaver.setUpComplexFindReplace()

Availability

Dreamweaver 3.0

Description

Prepares for an advanced text or tag search by loading the specified XML query.

Arguments

xmlQueryString

xmlQueryString is a string of XML code beginning with `<dwquery>` and ending with `</dwquery>`. (To get a string of the proper format, set up the query in the Find dialog box, click the Save Query button, open the query file in a text editor, and copy everything from the beginning of the `<dwquery>` tag to the end of the `</dwquery>` tag.)

Returns

Nothing.

Enabler

None.

Example

The first statement in the following code sets up a tag search and specifies that the scope of the search should be four files; the second statement performs the search and replace operation:

```
dw.setUpComplexFindReplace('<dwquery><queryparams
matchcase="false" ↵
ignorewhitespace="true" useregexp="false"/><find><qtag qname="a">↵
<qattribute qname="href" qcompare="=" qvalue="#"></qattribute>↵
<qattribute qname="onMouseOut" qcompare="=" qvalue="" ↵
qnegate="true"></qattribute></qtag></find><replace ↵
action="setAttribute" param1="onMouseOut" ↵
param2="this.style.color='#000000';↵
this.style.fontWeight='normal'"/></dwquery>');
dw.replaceAll();
```

`dreamweaver.setUpFind()`

Availability

Dreamweaver 3.0

Description

Prepares for a text or HTML source search by defining the search parameters for a subsequent `dw.findNext()` operation.

Arguments

searchObject

searchObject is an object for which the following properties can be defined:

- *searchString* is the text to search for.
- *searchSource* is a Boolean value indicating whether to search the HTML source.
- *{matchCase}* is a Boolean value indicating whether the search is case-sensitive. If this property is not explicitly set, it defaults to `false`.
- *{ignoreWhitespace}* is a Boolean value indicating whether white space differences should be ignored. *ignoreWhitespace* defaults to `false` if *useRegularExpressions* is `true` and `true` if *useRegularExpressions* is `false`.
- *{useRegularExpressions}* is a Boolean value indicating whether the *searchString* uses regular expressions. If this property is not explicitly set, it defaults to `false`.

Returns

Nothing.

Enabler

None.

Example

The following code demonstrates three ways to create a *searchObject* object:

```
var searchParams;  
searchParams.searchString = 'bgcolor="#FFCCFF";  
searchParams.searchSource = true;  
dw.setUpFind(searchParams);  
  
var searchParams = {searchString: 'bgcolor="#FFCCFF',  
searchSource: true};  
dw.setUpFind(searchParams);  
  
dw.setUpFind({searchString: 'bgcolor="#FFCCFF', searchSource: true});
```

`dreamweaver.setUpFindReplace()`

Availability

Dreamweaver 3.0

Description

Prepares for a text or HTML source search by defining the search parameters and the scope for a subsequent `dw.replace()` or `dw.replaceAll()` operation.

Arguments

searchObject

searchObject is an object for which the following properties can be defined:

- *searchString* is the text to search for.
- *replaceString* is the text with which to replace the selection.
- *searchSource* is a Boolean value indicating whether to search the HTML source.
- *{matchCase}* is a Boolean value indicating whether the search is case-sensitive. If this property is not explicitly set, it defaults to `false`.
- *{ignoreWhitespace}* is a Boolean value indicating whether white space differences should be ignored. *ignoreWhitespace* defaults to `false` if *useRegularExpressions* is `true` and `true` if *useRegularExpressions* is `false`.
- *{useRegularExpressions}* is a Boolean value indicating whether the *searchString* uses regular expressions. If this property is not explicitly set, it defaults to `false`.

Returns

Nothing.

Enabler

None.

Example

The following code demonstrates three ways to create a *searchObject* object:

```
var searchParams;  
searchParams.searchString = 'bgcolor="#FFCCFF";'  
searchParams.replaceString = 'bgcolor="#CCFFCC";'  
searchParams.searchSource = true;  
dw.setUpFindReplace(searchParams);  
  
var searchParams = {searchString: 'bgcolor="#FFCCFF',  
replaceString: 'bgcolor="#CCFFCC"', searchSource: true};  
dw.setUpFindReplace(searchParams);  
  
dw.setUpFindReplace({searchString: 'bgcolor="#FFCCFF',  
replaceString: 'bgcolor="#CCFFCC"', searchSource: true});
```

dreamweaver.showFindDialog()

Availability

Dreamweaver 3.0

Description

Opens the Find dialog box.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.canShowFindDialog()`

dreamweaver.showFindReplaceDialog()

Availability

Dreamweaver 3.0

Description

Opens the Replace dialog box.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.canShowFindDialog()`

Frame and frameset functions

Frame and frameset functions cover only two tasks: getting the names of the frames in a frameset and splitting a frame in two.

`dom.getFrameNames()`

Availability

Dreamweaver 3.0

Description

Gets a list of all the named frames in the frameset.

Arguments

None.

Returns

An array of strings, each a name of a frame in the current frameset. Any unnamed frames are skipped. If none of the frames in the frameset is named, an empty array is returned.

Enabler

None.

Example

For a document containing four frames, two of which are named, a call to `dw.getDocumentDOM().getFrameNames()` might return an array containing the following strings:

- "navframe"
- "main_content"

`dom.isDocumentInFrame()`

Availability

Dreamweaver 4.0

Description

Identifies whether or not the current document is being viewed inside a frameset.

Arguments

None.

Returns

Returns `true` if the document is in a frameset. Returns `false` otherwise.

Enabler

None.

dom.saveAllFrames()

Availability

Dreamweaver 4.0

Description

If the given document is a frameset or inside a frameset, saves all the frames and framesets from the same Document window. If the given document is not in a frameset, just saves the document. Opens the Save As dialog box for any documents that have not been previously saved.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.splitFrame()

Availability

Dreamweaver 3.0

Description

Splits the selected frame vertically or horizontally.

Arguments

splitDirection

splitDirection must be one of the following: "up", "down", "left", or "right".

Returns

Nothing.

Enabler

dom.canSplitFrame()

General editing functions

General editing functions handle common editing tasks that happen in the Document window. These functions insert text, HTML, and objects; apply, change, and remove font and character markup; modify tags and attributes; and more.

`dom.applyCharacterMarkup()`

Availability

Dreamweaver 3.0

Description

Applies the specified type of character markup to the selection. If the selection is an insertion point, applies the specified character markup to any subsequently typed text.

Arguments

tagName

tagName is the tag name associated with the character markup. It must be one of the following strings: "b", "cite", "code", "dfn", "em", "i", "kbd", "samp", "s", "strong", "tt", "u", or "var".

Returns

Nothing.

Enabler

None.

`dom.applyFontMarkup()`

Availability

Dreamweaver 3.0

Description

Applies the FONT tag and the specified attribute and value to the current selection.

Arguments

attribute, value

- *attribute* must be "face", "size", or "color".
- *value* is the value that should be assigned to the attribute; for example, "Arial, Helvetica, sans-serif", "5", or "#FF0000".

Returns

Nothing.

Enabler

None.

dom.deleteSelection()

Availability

Dreamweaver 3.0

Description

Deletes the selection in the document.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.editAttribute()

Availability

Dreamweaver 3.0

Description

Displays the appropriate interface for editing the specified attribute. In most cases, this is a dialog box. This function is valid only for the active document.

Arguments

attribute

Returns

Nothing.

Enabler

None.

dom.exitBlock()

Availability

Dreamweaver 3.0

Description

Exits the current paragraph or heading block, leaving the cursor outside of all block elements.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.getCharSet()

Availability

Dreamweaver 4.0

Description

Returns the charset attribute in the meta tag of the document.

Arguments

None.

Returns

The encoding identity of the document. So, for example, in *Latin1* document, the function will return `iso-8859-1`.

dom.getFontMarkup()

Availability

Dreamweaver 3.0

Description

Gets the value of the specified attribute of the FONT tag for the current selection.

Arguments

attribute

attribute must be "face", "size", or "color".

Returns

A string containing the value of the specified attribute, or an empty string if the attribute is not set.

Enabler

None.

dom.getLinkHref()**Availability**

Dreamweaver 3.0

Description

Gets the link that surrounds the current selection. This function is equivalent to looping through the parents and grandparents of the current node until a link is found, and then calling `getAttribute('HREF')` on the link.

Arguments

None.

Returns

A string containing the name of the linked file, expressed as a file:// URL.

Enabler

None.

dom.getLinkTarget()**Availability**

Dreamweaver 3.0

Description

Gets the target of the link that surrounds the current selection. This function is equivalent to looping through the parents and grandparents of the current node until a link is found, and then calling `getAttribute('TARGET')` on the link.

Arguments

None.

Returns

A string containing the value of the `TARGET` attribute for the link, or an empty string if no target is specified.

Enabler

None.

dom.getListTag()

Availability

Dreamweaver 3.0

Description

Gets the style of the selected list.

Arguments

None.

Returns

A string containing the tag associated with the list ("ul", "ol", or "dl"), or an empty string if no tag is associated with the list. This value is always returned in lowercase letters.

Enabler

None.

dom.getTextAlignment()

Availability

Dreamweaver 3.0

Description

Gets the alignment of the block that contains the selection.

Arguments

None.

Returns

A string containing the value of the ALIGN attribute for the tag associated with the block, or an empty string if the ALIGN attribute is not set for the tag. This value is always returned in lowercase letters.

Enabler

None.

dom.getTextFormat()

Availability

Dreamweaver 3.0

Description

Gets the block format of the selected text.

Arguments

None.

Returns

A string containing the block tag associated with the text (for example, "p", "h1", "pre", and so on, or an empty string if no block tag is associated with the selection). This value is always returned in lowercase letters.

Enabler

None.

dom.hasCharacterMarkup()

Availability

Dreamweaver 3.0

Description

Checks whether the selection already has the specified character markup.

Arguments

markupTagName

markupTagName is the name of the tag you're checking for. It must be one of the following strings: "b", "cite", "code", "dfn", "em", "i", "kbd", "samp", "s", "strong", "tt", "u", or "var".

Returns

A Boolean value indicating whether the entire selection has the specified character markup. The function returns `false` if only part of the selection has the specified markup.

Enabler

None.

dom.indent()

Availability

Dreamweaver 3.0

Description

Indents the selection using BLOCKQUOTE tags. If the selection is inside a list, indents the selection by nesting another list of the same type inside the current list.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.insertHTML()

Availability

Dreamweaver 3.0

Description

Inserts HTML content into the document at the current insertion point.

Arguments

contentToInsert, {*bReplaceCurrentSelection*}

- *contentToInsert* is the content you want to insert.
- *bReplaceCurrentSelection* is a Boolean value indicating whether the content should replace the current selection. If *bReplaceCurrentSelection* is false, the content is inserted after the current selection.

Returns

Nothing.

Enabler

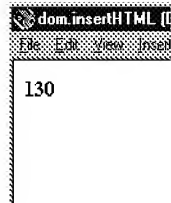
None.

Example

The following code inserts `130` into the current document:

```
var theDOM = dw.getDocumentDOM();  
theDOM.insertHTML('<b>130</b>');
```

This appears in the Document window as:



dom.insertObject()

Availability

Dreamweaver 3.0

Description

Inserts the specified object, prompting the user for parameters if necessary.

Arguments

objectName

objectName is the name of an object in the Configuration/Objects folder.

Returns

Nothing.

Enabler

None.

Example

A call to `dreamweaver.getDocumentDOM().insertObject('Button');` inserts a form button into the current document at the current insertion point or after the current selection.

Note: Although object files can be stored in separate folders, it's important that their file names be unique. If a file called `Button.htm` exists in the Forms folder and also in the MyObjects folder, Dreamweaver cannot distinguish between them.

dom.insertText()

Availability

Dreamweaver 3.0

Description

Inserts text content into the document at the current insertion point.

Arguments

contentToInsert, {*bReplaceCurrentSelection*}

- *contentToInsert* is the content you want to insert.
- *bReplaceCurrentSelection* is a Boolean value indicating whether the content should replace the current selection. If *bReplaceCurrentSelection* is false, the content is inserted after the current selection.

Returns

Nothing.

Enabler

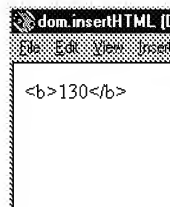
None.

Example

The following code inserts `<t;b>130` into the current document:

```
var theDOM = dw.getDocumentDOM();
theDOM.insertText('<b>130</b>');
```

This appears in the Document window as:



dom.newBlock()

Availability

Dreamweaver 3.0

Description

Creates a new block with the same tag and attributes as the block that contains the current selection, or creates a new paragraph if the cursor is outside of all blocks.

Arguments

None.

Returns

Nothing.

Enabler

None.

Example

If the current selection is inside a center-aligned paragraph, a call to `dreamweaver.getDocumentDOM().newBlock()` inserts `<p align="center">` after the current paragraph.

dom.notifyFlashObjectChanged()

Availability

Dreamweaver 4.0

Description

Notifies Dreamweaver that the current Flash object file has changed. Dreamweaver updates the Preview display, resizing it as necessary and preserving the width and height ratio from the natural size. For example, Flash Text uses this to update the text in the Layout view as the user changes its properties in the Command dialog box.

Arguments

None.

Returns

Nothing.

dom.outdent()

Availability

Dreamweaver 3.0

Description

Outdents the selection.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.removeCharacterMarkup()

Availability

Dreamweaver 3.0

Description

Removes the specified type of character markup from the selection.

Arguments

tagName

tagName is the tag name associated with the character markup. It must be one of the following strings: "b", "cite", "code", "dfn", "em", "i", "kbd", "samp", "s", "strong", "tt", "u", or "var".

Returns

Nothing.

Enabler

None.

dom.removeFontMarkup()

Availability

Dreamweaver 3.0

Description

Removes the specified attribute and its value from a FONT tag. If removing the attribute would leave only , then the FONT tag is also removed.

Arguments

attribute

attribute must be "face", "size", or "color".

Returns

Nothing.

Enabler

None.

dom.removeLink()

Availability

Dreamweaver 3.0

Description

Removes the hyperlink from the selection.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.resizeSelection()

Availability

Dreamweaver 3.0

Description

Resizes the selected object to the specified dimensions. To resize a layer or hotspot, use `dom.resizeSelectionBy()`.

Arguments

newWidth, newHeight

Returns

Nothing.

Enabler

None.

dom.setAttributeWithErrorChecking()

Availability

Dreamweaver 3.0

Description

Sets the specified attribute to the specified value for the current selection, prompting the user if the value is of the wrong type or if it is out of range. This function is valid only for the active document.

Arguments

attribute, value

Returns

Nothing.

Enabler

None.

dom.setLinkHref()

Availability

Dreamweaver 3.0

Description

Makes the selection a hyperlink or changes the value of the link that surrounds the current selection.

Arguments

linkHref

linkHref is the URL (document-relative path, root-relative path, or absolute URL) to link to. If the argument is omitted, the Select HTML File dialog box appears.

Returns

Nothing.

Enabler

dom.canSetLinkHref()

dom.setLinkTarget()

Availability

Dreamweaver 3.0

Description

Sets the target of the link that surrounds the current selection. This function is equivalent to looping through the parents and grandparents of the current node until a link is found, and then calling `setAttribute('TARGET')` on the link.

Arguments

{linkTarget}

linkTarget is a string representing a frame or window name, or one of the reserved targets ("_self", "_parent", "_top", or "_blank"). If the argument is omitted, the Set Target dialog box appears.

Returns

Nothing.

Enabler

None.

dom.setListBoxKind()

Availability

Dreamweaver 3.0

Description

Changes the kind of the selected **SELECT** menu.

Arguments

kind

kind must be either "menu" or "list box".

Returns

Nothing.

Enabler

None.

dom.showListPropertiesDialog()

Availability

Dreamweaver 3.0

Description

Opens the List Properties dialog box.

Arguments

None.

Returns

Nothing.

Enabler

dom.canShowListPropertiesDialog()

dom.setListTag()

Availability

Dreamweaver 3.0

Description

Sets the style of the selected list.

Arguments

listTag

listTag is the tag associated with the list. It must be "ol", "ul", "dl", or an empty string.

Returns

Nothing.

Enabler

None.

dom.setTextAlignment()**Availability**

Dreamweaver 3.0

Description

Sets the ALIGN attribute of the block that contains the selection to the specified value.

Arguments

alignValue

alignValue must be "left", "center", or "right".

Returns

Nothing.

Enabler

None.

dom.setTextFieldKind()**Availability**

Dreamweaver 3.0

Description

Sets the format of the selected text field.

Arguments

fieldType

fieldType must be "input", "textarea", or "password".

Returns

Nothing.

Enabler

None.

dom.showFontColorDialog()

Availability

Dreamweaver 3.0

Description

Opens the Color Picker dialog box.

Arguments

None.

Returns

Nothing.

Enabler

None.

dreamweaver.deleteSelection()

Availability

Dreamweaver 3.0

Description

Deletes the selection in the active document or the Site window; or, on the Macintosh, the edit field that has focus in a dialog box or floating panel.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.canDeleteSelection()`

dreamweaver.editFontList()

Availability

Dreamweaver 3.0

Description

Opens the Edit Font List dialog box.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.getFontList()`

Availability

Dreamweaver 3.0

Description

Gets a list of all the font groups that appear in the text Property inspector and in the Style Definition dialog box.

Arguments

None.

Returns

An array of strings representing each item in the font list.

Enabler

None.

Example

For the default installation of Dreamweaver, a call to `dw.getFontList()` would return an array containing the following items:

- "Arial, Helvetica, sans-serif"
- "Times New Roman, Times, serif"
- "Courier New, Courier, mono"
- "Georgia, Times New Roman, Times, serif"
- "Verdana, Arial, Helvetica, sans-serif"

`dreamweaver.getFontStyles()`

Availability

Dreamweaver 4.0

Description

Returns the styles that a specified TrueType font supports.

Arguments

fontName

fontName is a string containing the name of the font.

Returns

An array of three Boolean values indicating what the font supports. The first indicates whether or not the font supports *Bold*, the second *Italic*, and the third whether or not the font supports both *Bold and Italic* together.

`dreamweaver.getKeyState()`

Availability

Dreamweaver 3.0

Description

Determines whether the specified modifier key is down.

Arguments

key

key must be one of the following: "Cmd", "Ctrl", "Alt", or "Shift". In Windows, "Cmd" and "Ctrl" both refer to the Control key; on the Macintosh, "Alt" refers to the Option key.

Returns

A Boolean value indicating whether the key is down.

Enabler

None.

Example

The following code checks that both the Shift and Control keys (Windows) or Shift and Command keys (Macintosh) are down before performing an operation:

```
if (dw.getKeyState("Shift") && dw.getKeyState("Cmd")){  
    // execute code  
}
```

`dreamweaver.getSystemFontList()`

Availability

Dreamweaver 4.0

Description

Returns a list of fonts for the system. This function can get either all fonts or TrueType only. These fonts are needed for the Flash Text object.

Arguments

fontTypes

fontTypes is a string containing either "all" or "TrueType".

Returns

An array of strings containing all the font names; returns null if no fonts were found.

Global application functions

Global application functions act on the entire application. They provide such functionality as quitting and accessing preferences.

`dreamweaver.getShowDialogsOnInsert()`

Availability

Dreamweaver 3.0

Description

Checks whether the Show Dialog When Inserting Objects option is turned on in the General preferences.

Arguments

None.

Returns

A Boolean value indicating whether the option is on.

Enabler

None.

`dreamweaver.quitApplication()`

Availability

Dreamweaver 3.0

Description

Quits Dreamweaver after the script that calls this function finishes executing.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.showAboutBox()`

Availability

Dreamweaver 3.0

Description

Opens the About box.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.showPreferencesDialog()`

Availability

Dreamweaver 3.0

Description

Opens the Preferences dialog box.

Arguments

{whichTab}

The argument must be one of the following strings: "general", "external editors", "floaters", "fonts", "highlighting", "html colors", "html format", "html rewriting", "invisible elements", "layers", "browsers", "quick tag editor", "site ftp", "status bar", "css styles", and "translation". If Dreamweaver does not recognize the argument as a valid pane name, or if the argument is omitted, the dialog box opens to the last active pane.

Returns

Nothing.

Enabler

None.

Global document functions

Global document functions act on the entire document. They check spelling, check target browsers, set page properties, and determine correct object references for elements in the document.

`dom.checkSpelling()`

Availability

Dreamweaver 3.0

Description

Checks the spelling in the document, opening the Check Spelling dialog box if necessary, and notifies the user when the check is complete.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dom.checkTargetBrowsers()`

Availability

Dreamweaver 3.0

Description

Runs a target browser check on the document. To run a target browser check on a folder or group of files, use `site.checkTargetBrowsers()`.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dom.showPagePropertiesDialog()`

Availability

Dreamweaver 3.0

Description

Opens the Page Properties dialog box.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.getElementRef()`

Availability

Dreamweaver 2.0

Description

Gets the Netscape or IE object reference for a specific tag object in the DOM tree.

Arguments

NSorIE, *tagObject*

- *NSorIE* must be either "NS 4.0" or "IE 4.0". The DOM and rules for nested references differ in Navigator 4.0 and Internet Explorer 4.0. This argument specifies for which browser to return a valid reference.
- *tagObject* is a tag object in the DOM tree.

Returns

A string representing a valid JavaScript reference to the object, such as `document.layers['myLayer']`.

- Dreamweaver returns correct references for Internet Explorer for A, AREA, APPLET, EMBED, DIV, SPAN, INPUT, SELECT, OPTION, TEXTAREA, OBJECT, and IMG tags.
- Dreamweaver returns correct references for Navigator for A, AREA, APPLET, EMBED, LAYER, ILAYER, SELECT, OPTION, TEXTAREA, OBJECT, and IMG tags, and for absolutely positioned DIV and SPAN tags. For DIV and SPAN tags that are not absolutely positioned, Dreamweaver returns "cannot reference <tag>".
- Dreamweaver does not return references for unnamed objects. If an object does not contain either a NAME or an ID attribute, then Dreamweaver returns "unnamed <tag>". If the browser does not support a reference by name, Dreamweaver references the object by index (for example, `document.myform.applets[3]`).
- Dreamweaver returns references for named objects contained in unnamed forms and layers (for example, `document.forms[2].myCheckbox`).

Enabler

None.

History functions

History functions deal with undoing, redoing, recording, and playing steps that appear in the History panel (programmed into the API as "palette"). A step is any repeatable change to the document or to a selection in the document. Methods of the `dreamweaver.historyPalette` object either control or act on the selection in the History panel, not in the current document.

`dom.redo()`

Availability

Dreamweaver 3.0

Description

Redoes the step that was just undone in the document.

Arguments

None.

Returns

Nothing.

Enabler

`dom.canRedo()`

`dom.undo()`

Availability

Dreamweaver 3.0

Description

Undoes the previous step in the document.

Arguments

None.

Returns

Nothing.

Enabler

`dom.canUndo()`

`dreamweaver.getRedoText()`

Availability

Dreamweaver 3.0

Description

Gets the text associated with the editing operation that will be redone if the user chooses Edit > Redo or presses Ctrl+Y (Windows) or Command+Y (Macintosh).

Arguments

None.

Returns

A string containing the text associated with the editing operation that will be redone.

Enabler

None.

Example

If the user's last action was to make the selection bold, a call to `dw.getRedoText()` would return "Repeat Apply Bold".

`dreamweaver.getUndoText()`

Availability

Dreamweaver 3.0

Description

Gets the text associated with the editing operation that will be undone if the user chooses Edit > Undo or presses Ctrl+Z (Windows) or Command+Z (Macintosh).

Arguments

None.

Returns

A string containing the text associated with the editing operation that will be undone.

Enabler

None.

Example

If the user's last action was to apply a CSS style to a selected range of text, a call to `dw.getUndoText()` would return "Undo Apply ".

`dreamweaver.playRecordedCommand()`**Availability**

Dreamweaver 3.0

Description

Plays the recorded command in the active document.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.canPlayRecordedCommand()`

`dreamweaver.redo()`**Availability**

Dreamweaver 3.0

Description

Redoes the step that was just undone in the Document window, dialog box, floating panel, or Site window pane that has focus.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.canRedo()`

dreamweaver.startRecording()

Availability

Dreamweaver 3.0

Description

Starts recording steps in the active document. The previously recorded command is immediately discarded.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.isRecording()` (must return false)

dreamweaver.stopRecording()

Availability

Dreamweaver 3.0

Description

Stops recording without prompting the user.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.isRecording()` (must return true)

dreamweaver.undo()

Availability

Dreamweaver 3.0

Description

Undoes the previous step in the Document window, dialog box, floating panel, or Site window pane that has focus.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.canUndo()`

`dreamweaver.historyPalette.clearSteps()`**Availability**

Dreamweaver 3.0

Description

Clears all steps from the History panel, and disables the Undo and Redo menu items.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.historyPalette.copySteps()`**Availability**

Dreamweaver 3.0

Description

Copies the specified history steps to the clipboard. Dreamweaver warns the user of possible unintended consequences if the specified steps include an unrepeatable action.

Arguments

arrayOfIndices

arrayOfIndices is an array of position indices in the History panel.

Returns

A string containing the JavaScript that corresponds to the specified history steps.

Enabler

None.

Example

The following code copies the first four steps in the History panel:

```
dw.historyPalette.copySteps([0,1,2,3]);
```

`dreamweaver.historyPalette.getSelectedSteps()`

Availability

Dreamweaver 3.0

Description

Determines which portion of the History panel is selected.

Arguments

None.

Returns

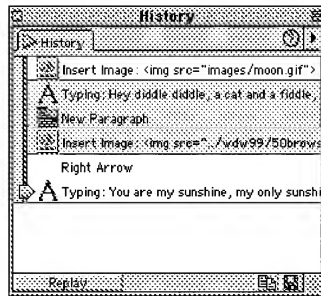
An array containing the position indices of all the selected steps.

Enabler

None.

Example

If the second, third, and fourth steps are selected in the History panel, as shown, a call to `dw.historyPalette.getSelectedSteps()` would return `[1,2,3]`.



`dreamweaver.historyPalette.getStepCount()`

Availability

Dreamweaver 3.0

Description

Gets the number of steps in the History panel.

Arguments

None.

Returns

An integer representing the number of steps that are currently listed in the History panel.

Enabler

None.

dreamweaver.historyPalette.getStepsAsJavaScript()

Availability

Dreamweaver 3.0

Description

Gets the JavaScript equivalent of the specified history steps.

Arguments

arrayOfIndices

arrayOfIndices is an array of position indices in the History panel.

Returns

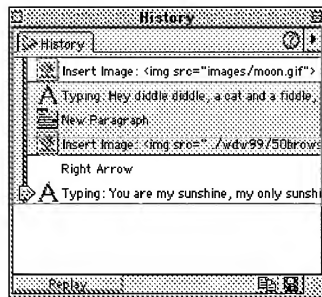
A string containing the JavaScript that corresponds to the specified history steps.

Enabler

None.

Example

If the three steps shown in this example are selected in the History panel, a call to `dw.historyPalette.getStepsAsJavaScript(dw.historyPalette.getSelectedSteps())` returns `"dw.getDocumentDOM().insertText('Hey diddle diddle, a cat and a fiddle, the cow jumped over the moon.');"dw.getDocumentDOM().newBlock();\n dw.getDocumentDOM().insertHTML('', true);\n"`:



`dreamweaver.historyPalette.getUndoState()`

Availability

Dreamweaver 3.0

Description

Gets the current undo state.

Arguments

None.

Returns

The position of the Undo marker in the History panel.

Enabler

None.

`dreamweaver.historyPalette.replaySteps()`

Availability

Dreamweaver 3.0

Description

Replays the specified history steps in the active document. Dreamweaver warns the user of possible unintended consequences if the specified steps include an unrepeatable action.

Arguments

arrayOfIndices

arrayOfIndices is an array of position indices in the History panel.

Returns

A string containing the JavaScript that corresponds to the specified history steps.

Enabler

None.

Example

A call to `dw.historyPalette.replaySteps([0,2,3])` plays the first, third, and fourth steps in the History panel.

`dreamweaver.historyPalette.saveAsCommand()`

Availability

Dreamweaver 3.0

Description

Opens the Save As Command dialog box, which lets the user save the specified steps as a command. Dreamweaver warns the user of possible unintended consequences if the steps include an unrepeatable action.

Arguments

arrayOfIndices

arrayOfIndices is an array of position indexes in the History panel.

Returns

A string containing the JavaScript that corresponds to the specified history steps.

Enabler

None.

Example

The following code saves the fourth, sixth, and eighth steps in the History panel as a command:

```
dw.historyPalette.saveAsCommand([3,5,7]);
```

`dreamweaver.historyPalette.setSelectedSteps()`

Availability

Dreamweaver 3.0

Description

Selects the specified steps in the History panel.

Arguments

arrayOfIndices

arrayOfIndices is an array of position indices in the History panel. If no argument is supplied, all steps are deselected.

Returns

None.

Enabler

None.

Example

The following code selects the first, second, and third steps in the History panel:

```
dw.historyPalette.setSelectedSteps([0,1,2]);
```

`dreamweaver.historyPalette.setUndoState()`

Availability

Dreamweaver 3.0

Description

Performs the correct number of undo or redo operations to arrive at the specified undo state.

Arguments

undoState

undoState is the object returned by `dw.historyPalette.getUndoState()`.

Returns

Nothing.

Enabler

None.

HTML style functions

HTML style functions handle applying, creating, and deleting HTML styles. Methods of the `dreamweaver.htmlStylePalette` object either control or act on the selection in the HTML Styles panel (programmed into the API as “palette”), not in the current document.

`dom.applyHTMLStyle()`

Availability

Dreamweaver 3.0

Description

Applies the specified HTML style to the current selection. This function is valid only for the active document.

Arguments

htmlStyleName

Returns

Nothing.

Enabler

None.

`dreamweaver.htmlStylePalette.deleteSelectedStyle()`

Availability

Dreamweaver 3.0

Description

Removes the selected style from the HTML Styles panel.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.htmlStylePalette.canEditSelection()`

`dreamweaver.htmlStylePalette.duplicateSelectedStyle()`

Availability

Dreamweaver 3.0

Description

Duplicates the selected style and opens the Define HTML Style dialog box.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.htmlStylePalette.canEditSelection()`

`dreamweaver.htmlStylePalette.editSelectedStyle()`

Availability

Dreamweaver 3.0

Description

Opens the Define HTML Style dialog box for the selected style.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.htmlStylePalette.canEditSelection()`

dreamweaver.htmlStylePalette.getSelectedStyle()

Availability

Dreamweaver 3.0

Description

Gets the name of the selected style in the HTML Style panel.

Arguments

None.

Returns

A string containing the name of the selected style.

Enabler

None.

dreamweaver.htmlStylePalette.getStyles()

Availability

Dreamweaver 3.0

Description

Gets a list of all of the names of the defined HTML styles.

Arguments

None.

Returns

An array of strings, each representing the name of an HTML style. If no HTML styles are defined, an empty array is returned.

Enabler

None.

`dreamweaver.htmlStylePalette.newStyle()`

Availability

Dreamweaver 3.0

Description

Opens the Define HTML Style dialog box for a new, untitled style.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.htmlStylePalette.setSelectedStyle()`

Availability

Dreamweaver 3.0

Description

Selects the specified style in the HTML Style panel.

Arguments

htmlStyleName

Returns

Nothing.

Enabler

None.

JavaScript Debugger functions

These commands customize the Dreamweaver JavaScript Debugger behavior. For more information about the Dreamweaver JavaScript Debugger, see “JavaScript Debugger Modules” on page 61.

`dom.instrumentDocument ()`

Availability

Dreamweaver 4.0

Description

Creates the debug version of the document and any external .js files it references. This function parses the JavaScript in the document and calls the *debuggerModule* for code snippets to insert at various points in the JavaScript file. The *debuggerModule* is also notified of syntax errors and warnings. This function fails under any of the following conditions: syntax errors, a file error, or the document cannot be debugged for some reason. Temporary files are never deleted immediately, even if the function fails.

Arguments

debuggerModule, *outputFileName*

- *debuggerModule* is the name of a special Dreamweaver module file that implements the instrumentation API. The module is located in the Configuration/Debugger folder of the Dreamweaver Program Files directory.
- *outputFileName* is optional; it is the name to use for the debug version of the .htm file. If omitted, a temporary file is created. The temporary file is deleted when Dreamweaver exits. If the specified *outputFileName* exists, the existing file is replaced. The file is always written in the same directory as the source document, so it cannot have the same name as the source document. If a path is specified, it is ignored. The debug version of externally referenced .js files is named by adding *outputFileName* to the beginning of the original file name of the .js file.

Returns

An array of pairs of file URLs. Each pair consists of the URL of the original source file followed by the URL of the debug version that was created. The first pair is always the .htm files and any subsequent entries are .js files referenced by the .htm file. If the function fails, then `null` is returned. Note that a pair of URLs is actually two entries in the array. So, if `returnValue = dom.instrumentDocument(test.htm)`, then `returnValue[0]` is the URL of `test.htm` and `returnValue[1]` is the URL of the debug version of `test.htm`.

`dreamweaver.debugDocument()`

Availability

Dreamweaver 4.0

Description

Creates the debug version of the current document and starts the debugger. This function can be used only with one of the browsers for which Dreamweaver supports JavaScript debugging (see “`dreamweaver.getDebugBrowserList()`” on page 305). This function does not prompt the user if it cannot determine the browser type. If syntax errors or warnings occur, a Results window is opened to display the messages. If no errors occur, the debug version of the HTML document appears in the specified browser. If warnings occur, but no errors, the Results window appears and debugging begins.

The creation of the debug version is implemented with a call to `dom.instrumentDocument()` using the one of the default instrumentation modules. The debug version of the document is temporary and is deleted the next time the debugger starts or when Dreamweaver exits.

Arguments

fileName, [*browserName*]

- *fileName* is the name of the file to be opened, expressed as an absolute URL.
- *browserName* is optional; it specifies the name of the target browser as defined in the Preview settings in Browser Preferences. It can also be ‘primary’ or ‘secondary’. If omitted, the primary browser is used by default.

Returns

Nothing.

`dreamweaver.getDebugBrowserList()`

Availability

Dreamweaver 4.0

Description

Returns the defined browsers for which Dreamweaver supports JavaScript debugging. For Windows, Dreamweaver supports debugging only in Internet Explorer 5.0 and later and Netscape 4.5 and later. For Macintosh, Dreamweaver supports debugging only in Netscape 4.5 and later.

Arguments

None.

Returns

An array of browser names in the same format as that returned by `getBrowserList()`.

`dreamweaver.getIsAnyBreakpoints()`

Availability

Dreamweaver 4.0

Description

Finds if any breakpoints are set in any files.

Arguments

None.

Returns

`bBreakpointsSet`

Boolean `true` if any breakpoints are set in any files.

`dreamweaver.removeAllBreakpoints()`

Availability

Dreamweaver 4.0

Description

Removes all breakpoints in all files.

Arguments

None.

Returns

Nothing.

dreamweaver.startDebugger()

Availability

Dreamweaver 4.0

Description

Opens the Debugger window with original source .htm file and .js files listed in `sourceFileList()`, then launches the browser with the debug version file specified by `debugFileName()`. This function does not prompt the user if it cannot determine the browser type.

The creation of the debug version is accomplished with a call to `dom.instrumentDocument()`.

Arguments

debugFileName, *sourceFileList*, *isTempFiles*, {*browserName*}

- *debugFileName* is the name of a debug version file to launch in the browser.
- *sourceFileList* is an array of URL pairs comprising the source file and the instrumented file; the first pair is the .htm file and subsequent pairs are the external .js files. Each URL is expressed as an absolute file URL.
- *isTempFiles* is a Boolean indicating whether the instrumented files should be tracked and deleted the next time the debugger is launched, or when Dreamweaver shuts down.
- *browserName* is optional; it specifies the name of the target browser as defined in the Preview settings in Browser Preferences. It can also be 'primary' or 'secondary'. If omitted, the primary browser is used by default.

Returns

Nothing.

Keyboard functions

Keyboard functions mimic document navigation tasks that are accomplished by pressing the arrow, Backspace, Delete, Page Up, and Page Down keys. In addition to such general arrow and key methods as `arrowLeft()` and `backspaceKey()`, Dreamweaver also provides methods for moving to the next or previous word or paragraph, and moving to the end of line or document or start of the line or document.

`dom.arrowDown()`

Availability

Dreamweaver 3.0

Description

Moves the insertion point down the specified number of times.

Arguments

{nTimes}, {bShiftIsDown}

- *nTimes* is the number of times the insertion point should be moved down. If this argument is omitted, the default is 1.
- *bShiftIsDown* is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

Enabler

None.

`dom.arrowLeft()`

Availability

Dreamweaver 3.0

Description

Moves the insertion point left the specified number of times.

Arguments

{nTimes}, {bShiftIsDown}

- *nTimes* is the number of times the insertion point should be moved left. If this argument is omitted, the default is 1.
- *bShiftIsDown* is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

Enabler

None.

dom.arrowRight()**Availability**

Dreamweaver 3.0

Description

Moves the insertion point right the specified number of times.

Arguments

{nTimes}, {bShiftIsDown}

- *nTimes* is the number of times the insertion point should be moved right. If this argument is omitted, the default is 1.
- *bShiftIsDown* is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

Enabler

None.

dom.arrowUp()**Availability**

Dreamweaver 3.0

Description

Moves the insertion point up the specified number of times.

Arguments

{nTimes}, {bShiftIsDown}

- *nTimes* is the number of times the insertion point should be moved up. If this argument is omitted, the default is 1.
- *bShiftIsDown* is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

Enabler

None.

dom.backspaceKey()

Availability

Dreamweaver 3.0

Description

Equivalent to pressing the Backspace key the specified number of times. The exact behavior differs depending on whether there is a current selection or just an insertion point.

Arguments

{nTimes}

nTimes is the number of times a Backspace operation should occur. If omitted, the default is 1.

Returns

Nothing.

Enabler

None.

dom.deleteKey()

Availability

Dreamweaver 3.0

Description

Equivalent to pressing the Delete key the specified number of times. The exact behavior differs depending on whether there is a current selection or just an insertion point.

Arguments

{nTimes}

nTimes is the number of times a Delete operation should occur. If omitted, the default is 1.

Returns

Nothing.

Enabler

None.

dom.endOfDocument()

Availability

Dreamweaver 3.0

Description

Moves the insertion point to the end of the document (that is, after the last visible content in the Document window, or after the closing HTML tag in the Code inspector, depending on which window has focus).

Arguments

{bShiftIsDown}

bShiftIsDown is a Boolean value indicating whether to extend the selection. If omitted, the default is `false`.

Returns

Nothing.

Enabler

None.

dom.endOfLine()

Availability

Dreamweaver 3.0

Description

Moves the insertion point to the end of the line.

Arguments

{bShiftIsDown}

bShiftIsDown is a Boolean value indicating whether to extend the selection. If omitted, the default is `false`.

Returns

Nothing.

Enabler

None.

dom.nextParagraph()

Availability

Dreamweaver 3.0

Description

Moves the insertion point to the beginning of the next paragraph (or skips multiple paragraphs if *nTimes* is greater than 1).

Arguments

{nTimes}, *{bShiftIsDown}*

- *nTimes* is the number of paragraphs ahead the insertion point should jump. If this argument is omitted, the default is 1.
- *bShiftIsDown* is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

Enabler

None.

dom.nextWord()

Availability

Dreamweaver 3.0

Description

Moves the insertion point to the beginning of the next word (or skips multiple words if *nTimes* is greater than 1).

Arguments

{nTimes}, *{bShiftIsDown}*

- *nTimes* is the number of words ahead the insertion point should jump. If this argument is omitted, the default is 1.
- *bShiftIsDown* is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

Enabler

None.

dom.pageDown()

Availability

Dreamweaver 3.0

Description

Moves the insertion point down one page (equivalent to pressing Page Down).

Arguments

{nTimes}, {bShiftIsDown}

- *nTimes* is the number of pages down the insertion point should move. If this argument is omitted, the default is 1.
- *bShiftIsDown* is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

Enabler

None.

dom.pageUp()

Availability

Dreamweaver 3.0

Description

Moves the insertion point up one page (equivalent to pressing Page Up).

Arguments

{nTimes}, {bShiftIsDown}

- *nTimes* is the number of pages up the insertion point should move. If this argument is omitted, the default is 1.
- *bShiftIsDown* is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is `false`.

Returns

Nothing.

Enabler

None.

dom.previousParagraph()

Availability

Dreamweaver 3.0

Description

Moves the insertion point to the beginning of the previous paragraph (or skips multiple paragraphs if *nTimes* is greater than 1).

Arguments

{nTimes}, *{bShiftIsDown}*

- *nTimes* is the number of paragraphs back the insertion point should jump. If this argument is omitted, the default is 1.
- *bShiftIsDown* is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is *false*.

Returns

Nothing.

Enabler

None.

dom.previousWord()

Availability

Dreamweaver 3.0

Description

Moves the insertion point to the beginning of the previous word (or skips multiple words if *nTimes* is greater than 1).

Arguments

{nTimes}, *{bShiftIsDown}*

- *nTimes* is the number of words back the insertion point should jump. If this argument is omitted, the default is 1.
- *bShiftIsDown* is a Boolean value indicating whether to extend the selection. If this argument is omitted, the default is *false*.

Returns

Nothing.

Enabler

None.

dom.startOfDocument()

Availability

Dreamweaver 3.0

Description

Moves the insertion point to the beginning of the document (that is, before the first visible content in the Document window, or before the opening HTML tag in the Code inspector, depending on which window has focus).

Arguments

{bShiftIsDown}

bShiftIsDown is a Boolean value indicating whether to extend the selection. If omitted, the default is `false`.

Returns

Nothing.

Enabler

None.

dom.startOfLine()

Availability

Dreamweaver 3.0

Description

Moves the insertion point to the beginning of the line.

Arguments

{bShiftIsDown}

bShiftIsDown is a Boolean value indicating whether to extend the selection. If omitted, the default is `false`.

Returns

Nothing.

Enabler

None.

mapKeyCodeToChar()

Description

Takes a key code as retrieved from the event object's `keyCode` field and translates it to a character. You should first check to see if the key code is a special key, such as HOME, PGUP, and so on. If the key code is not a special key, then this method should be used to translate it to a character code suitable for display to the user.

Arguments

keyCode

keyCode is the key code to translate to a character.

Returns

Nothing.

Layer and image map functions

Layer and image map functions handle aligning, resizing, and moving layers and image map hotspots. The function description indicates if it applies to layers or to hotspots.

dom.align()

Availability

Dreamweaver 3.0

Description

Aligns the selected layers or hotspots left, right, top, or bottom.

Arguments

alignDirection

alignDirection is the edge on which the layers or hotspots should be aligned—"left", "right", "top", or "bottom".

Returns

Nothing.

Enabler

`dom.canAlign()`

`dom.arrange()`

Availability

Dreamweaver 3.0

Description

Moves the selected hotspots in the specified direction.

Arguments

toBackOrFront

toBackOrFront is the direction in which the hotspots should be moved—"front" or "back".

Returns

Nothing.

Enabler

`dom.canArrange()`

`dom.makeSizesEqual()`

Availability

Dreamweaver 3.0

Description

Makes the selected layers or hotspots equal in height, width, or both. The last layer or hotspot selected is the guide.

Arguments

bHoriz, *bVert*

- *bHoriz* is a Boolean value indicating whether to resize the layers or hotspots horizontally.
- *bVert* is a Boolean value indicating whether to resize the layers or hotspots vertically.

Returns

Nothing.

Enabler

None.

dom.moveSelectionBy()

Availability

Dreamweaver 3.0

Description

Moves the selected layers or hotspots by the specified number of pixels horizontally and vertically.

Arguments

x, y

- *x* is the number of pixels the selection should be moved horizontally.
- *y* is the number of pixels the selection should be moved vertically.

Returns

Nothing.

Enabler

None.

dom.resizeSelectionBy()

Availability

Dreamweaver 3.0

Description

Resizes the currently selected layer or hotspot.

Arguments

left, top, bottom, right

- *left* is the new position of the left boundary of the layer or hotspot.
- *top* is the new position of the top boundary of the layer or hotspot.
- *bottom* is the new position of the bottom boundary of the layer or hotspot.
- *right* is the new position of the right boundary of the layer or hotspot.

Returns

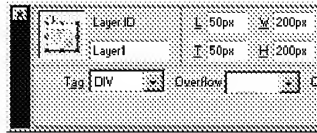
Nothing.

Enabler

None.

Example

If the selected layer has the Left, Top, Width, and Height properties shown, calling `dw.getDocumentDOM().resizeSelectionBy(-10,-30,30,10)` would be equivalent to resetting Left to 40, Top to 20, Width to 240, and Height to 240.



`dom.setLayerTag()`

Availability

Dreamweaver 3.0

Description

Specifies the HTML tag that defines the selected layer or layers.

Arguments

tagName

tagName must be "layer", "ilayer", "div", or "span".

Returns

Nothing.

Enabler

None.

Library and template functions

Library and template functions manage operations related to library items and templates, such as creating, updating, and breaking links between a document and a template or library item. Methods of the `dreamweaver.libraryPalette` object either control or act on the selection in the Library panel (programmed into the API as “palette”), not in the current document. Likewise, methods of the `dreamweaver.templatePalette` object control or act on the selection in the Template panel.

`dom.applyTemplate()`

Availability

Dreamweaver 3.0

Description

Applies a template to the current document. If no argument is supplied, the Select Template dialog box appears. This function is valid only for the active document.

Arguments

[templateURL], bMaintainLink

- *templateURL* is the path to a template in the current site, expressed as a file:// URL.
- *bMaintainLink* is a Boolean indicating whether to maintain the link to the original template (true) or not (false).

Returns

Nothing.

Enabler

`dom.canApplyTemplate()`

dom.detachFromLibrary()

Availability

Dreamweaver 3.0

Description

Detaches the selected instance of a library item from its associated LBI file by removing the locking tags from around the selection. This function is equivalent to clicking Detach from Original in the Property inspector.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.detachFromTemplate()

Availability

Dreamweaver 3.0

Description

Detaches the current document from its associated template.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.getAttachedTemplate()

Availability

Dreamweaver 3.0

Description

Gets the path of the template that is associated with the document.

Arguments

None.

Returns

A string containing the path of the template, expressed as a file:// URL.

Enabler

None.

dom.getEditableRegionList()

Availability

Dreamweaver 3.0

Description

Gets a list of all the editable regions in the body of the document.

Arguments

None.

Returns

An array of element nodes.

Enabler

None.

Example

See “dom.getSelectedEditableRegion()” on page 324.

dom.getIsLibraryDocument()

Availability

Dreamweaver 3.0

Description

Determines whether the document is a library item.

Arguments

None.

Returns

A Boolean value indicating whether the document is an LBI file.

Enabler

None.

dom.getIsTemplateDocument()

Availability

Dreamweaver 3.0

Description

Determines whether the document is a template.

Arguments

None.

Returns

A Boolean value indicating whether the document is a DWT file.

Enabler

None.

dom.getSelectedEditableRegion()

Availability

Dreamweaver 3.0

Description

If the selection or insertion point is inside an editable region, gets the position of the editable region among all others in the body of the document.

Arguments

None.

Returns

An index into the array returned by `dom.getEditableRegionList()`.

Enabler

None.

Example

The following code shows a dialog box containing the contents of the selected editable region:

```
var theDOM = dw.getDocumentDOM();
var edRegs = theDOM.getEditableRegionList();
var selReg = theDOM.getSelectedEditableRegion();
alert(edRegs[selReg].innerHTML);
```

dom.insertLibraryItem()

Availability

Dreamweaver 3.0

Description

Inserts an instance of a library item into the document.

Arguments

libraryItemURL

libraryItemURL is the path to an LBI file, expressed as a file:// URL.

Returns

Nothing.

Enabler

None.

dom.markSelectionAsEditable()

Availability

Dreamweaver 3.0

Description

Displays the New Editable Region dialog box. When the user clicks New Region, Dreamweaver marks the selection as editable and leaves any text as is.

Arguments

None.

Returns

Nothing.

Enabler

dom.canMarkSelectionAsEditable()

dom.newEditableRegion()

Availability

Dreamweaver 3.0

Description

Displays the New Editable Region dialog box. When the user clicks New Region, Dreamweaver inserts the name of the region, surrounded by braces ({ }), into the document at the insertion point location.

Arguments

None.

Returns

Nothing.

Enabler

dom.canMakeNewEditableRegion()

dom.removeEditableRegion()

Availability

Dreamweaver 3.0

Description

Removes an editable region from the document. If the editable region contains any content, the content is preserved—only the editable region markers are removed.

Arguments

None.

Returns

Nothing.

Enabler

dom.canRemoveEditableRegion()

dom.updateCurrentPage()

Availability

Dreamweaver 3.0

Description

Updates the document's library items, templates, or both. This function is valid only for the active document.

Arguments

{typeOfUpdate}

typeOfUpdate, if supplied, must be "library", "template", or "both". If omitted, the default is "both".

Returns

Nothing.

Enabler

None.

`dreamweaver.updatePages()`

Availability

Dreamweaver 3.0

Description

Opens the Update Pages dialog box and selects the specified options.

Arguments

{typeOfUpdate}

typeOfUpdate must be "library", "template", or "both". If the argument is omitted, it defaults to "both".

Returns

Nothing.

Enabler

None.

`dreamweaver.templatePalette.newBlankTemplate()`

Availability

Dreamweaver 3.0

Description

Creates a new template.

Arguments

None.

Returns

Nothing.

Enabler

None.

Menu functions

Menu functions deal with optimizing and reloading the menus in Dreamweaver. The `dreamweaver getMenuNeedsUpdating()` and `dreamweaver notifyMenuUpdated()` functions are designed specifically to prevent unnecessary update routines from running on the dynamic menus that are built into Dreamweaver.

`dreamweaver.getMenuNeedsUpdating()`

Availability

Dreamweaver 3.0

Description

Checks whether the specified menu needs to be updated.

Arguments

menuId

menuId is a string containing the value of the `id` attribute for the menu item (as specified in the `menus.xml` file).

Returns

A Boolean value indicating whether the menu needs to be updated. This function returns `false` only if `dreamweaver.notifyMenuUpdated()` has been called with this *menuId*, and the return value of *menuListFunction* has not changed since that time. See `dreamweaver.notifyMenuUpdated()` for more information.

Enabler

None.

dreamweaver.notifyMenuUpdated()

Availability

Dreamweaver 3.0

Description

Notifies Dreamweaver when the specified menu needs to be updated.

Arguments

menuId, *menuListFunction*

- *menuId* is a string containing the value of the `id` attribute for the menu item (as specified in the `menus.xml` file).
- *menuListFunction* must be one of the following strings:
 - "dw.cssStylePalette.getStyles()",
 - "dw.getDocumentDOM().getFrameNames()",
 - "dw.getDocumentDOM().getEditableRegionList",
 - "dw.getBrowserList()", "dw.getRecentFileList()",
 - "dw.getTranslatorList()", "dw.getFontList()",
 - "dw.getDocumentList()", "dw.htmlStylePalette.getStyles()", or
 - "site.getSites()".

Returns

Nothing.

Enabler

None.

dreamweaver.reloadMenus()

Availability

Dreamweaver 3.0

Description

Reloads the entire menu structure from the `menus.xml` file in the Configuration folder.

Arguments

None.

Returns

Nothing.

Enabler

None.

Path functions

Path functions get and manipulate the paths to various files and folders on the user's hard disk. These functions determine the path to the root of the site in which the current document resides, convert relative paths to absolute URLs, and more.

`dreamweaver.getConfigurationPath()`

Availability

Dreamweaver 2.0

Description

Gets the path to the Configuration folder, expressed as a file:// URL.

Arguments

None.

Returns

A string containing the path to the Configuration folder.

Enabler

None.

Example

This function is useful when referencing other extension files, which are all stored in the Configuration folder inside the Dreamweaver application folder. For example:

```
var sortCmd = dreamweaver.getConfigurationPath() + "/Commands/Sort Table.htm"
var sortDOM = dreamweaver.getDocumentDOM(sortCmd);
```

`dreamweaver.getDocumentPath()`

Availability

Dreamweaver1.2

Description

Gets the path of the specified document, expressed as a file:// URL. This function is equivalent to calling `dreamweaver.getDocumentDOM()` and then reading the `URL` property of the return value.

Arguments

sourceDoc

sourceDoc must be "document", "parent", "parent.frames[number]", or "parent.frames['frameName']". `document` specifies the document that has the focus and contains the current selection. "parent" specifies the parent frameset (if the currently selected document is in a frame), and "parent.frames[number]" and "parent.frames['frameName']" specify a document that is in a particular frame within the frameset containing the current document.

Returns

One of the following values:

- A string containing the URL of the specified document if the file was saved.
- An empty string if the file was not saved.

Enabler

None.

`dreamweaver.getSiteRoot()`

Availability

Dreamweaver 1.2

Description

Gets the local root folder (as specified in the Site Definition dialog box) for the site associated with the currently selected document, expressed as a file:// URL.

Arguments

None.

Returns

One of the following values:

- A string containing the URL of the local root folder of the site within which the file was saved.
- An empty string if the file is not associated with a site.

Enabler

None.

`dreamweaver.relativeToAbsoluteURL()`

Availability

Dreamweaver 2.0

Description

Given a relative URL and a point of reference (either the path to the current document or the site root), converts the relative URL to an absolute (file://) URL.

Arguments

reURL, *docPath*, *siteRoot*

- *reURL* is the URL to be converted.
- *docPath* is the path to a document on the user's disk (for example, the current document), expressed as a file:// URL or an empty string if *reURL* is a root-relative URL.
- *siteRoot* is the path to the site root, expressed as a file:// URL or an empty string if *reURL* is a document-relative URL.

Returns

An absolute URL string. The return value is generated as follows:

- If *relURL* is an absolute URL, no conversion takes place, and the return value is the same as *relURL*.
- If *relURL* is a document-relative URL, the return value is the combination of *docPath* + *relURL*.
- If *relURL* is a root-relative URL, the return value is the combination of *siteRoot* + *relURL*.

Enabler

None.

Quick Tag Editor functions

Quick Tag Editor functions navigate through the tags within and surrounding the current selection. They remove any tag in the hierarchy, wrap the selection inside a new tag, and show the Quick Tag Editor to let the user edit specific attributes for the tag.

dom.selectChild()**Availability**

Dreamweaver 3.0

Description

Selects a child of the current selection. Calling this function is equivalent to selecting the next tag to the right in the tag selector at the bottom of the Document window.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.selectParent()

Availability

Dreamweaver 3.0

Description

Selects the parent of the current selection. Calling this function is equivalent to selecting the next tag to the left in the tag selector at the bottom of the Document window.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.stripTag()

Availability

Dreamweaver 3.0

Description

Removes the tag from around the current selection, leaving the contents, if any. If the selection contains more than one tag or no tags, Dreamweaver reports an error.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.wrapTag()

Availability

Dreamweaver 3.0

Description

Wraps the specified tag around the current selection. If the selection is unbalanced, Dreamweaver reports an error.

Arguments

startTag

startTag is the source associated with the opening tag.

Returns

Nothing.

Enabler

None.

Example

The following code wraps a link around the current selection:

```
var theDOM = dw.getDocumentDOM();
var theSel = theDOM.getSelectedNode();
if (theSel.nodeType == Node.TEXT_NODE){
    theDOM.wrapTag('<a href="foo.html">');
}
```

dreamweaver.showQuickTagEditor()

Availability

Dreamweaver 3.0

Description

Displays the Quick Tag Editor for the current selection.

Arguments

{nearWhat}, {mode}

- *nearWhat*, if specified, must be either "selection" or "tag selector". The default value if this argument is omitted is "selection".
- *mode*, if specified, must be "default", "wrap", "insert", or "edit". If *mode* is "default" or omitted, Dreamweaver uses heuristics to determine the mode to use for the current selection. *mode* is ignored if *nearWhat* is "tag selector".

Returns

Nothing.

Enabler

None.

Report functions

Report functions provide access to the Dreamweaver reporting features so you can initiate, monitor and customize the reporting process. For more information, see “Reports” on page 55.

`dreamweaver.isReporting()`

Availability

Dreamweaver 4.0

Description

Checks to see if a reporting process is currently running.

Arguments

None.

Returns

Boolean value indicating whether a process is running (`true`) or not (`false`).

`dreamweaver.showReportsDialog()`

Availability

Dreamweaver 4.0

Description

Opens the Reports dialog box.

Arguments

None.

Returns

Nothing.

`dreamweaver.results.setResultData()`

Availability

Dreamweaver 4.0

Description

Adds new entries to the results window that is active during the reporting process. If a report is not being generated, then this function will have no effect.

Arguments

strFilePath, *strIcon*, *strDisplay*, *strDesc*, (*iLineNo*), (*iStartSel*), (*iEndSel*)

- *strFilePath* is a fully qualified URL to the file.
- *strIcon* is the fully qualified path to the icon to use for the results entry. You can reference built-in icons by specifying a number between 1 and 10.
- *strDisplay* is a string to display in the first column; this is usually a file name.
- *strDesc* is a description to go along with the entry.
- *iLineNo* is the number of lines in the file, if specified.
- *iStartSel* is the start of the offset into the file, if specified.
- *iEndSel* is the end of offset into the file; this is required only if *iStartSel* is specified.

Returns

Nothing.

Results window functions

Results window functions allow you to create a window displaying formatted data. Use these functions to create custom windows similar to the output from the JavaScript Debugger window or the Report Results window.

`dreamweaver.createResultsWindow()`

Availability

Dreamweaver 4.0

Description

Creates a new Results window and returns a JavaScript object reference to the window.

Arguments

strName, *arrColumns*

- *strName* is the string to use for the window's title.
- *arrColumns* is an array of column names to use in the list control.

Returns

An object reference to the created window.

`resWin.addItem()`

Availability

Dreamweaver 4.0

Description

Adds a new item to the Results window.

Arguments

strIcon, *strDesc*, *iStartSel*, *iEndSel*, *colNdata*

- *strIcon* is an icon to display. Specify 0 for no icon.
- *strDesc* is a detailed description of code font item. Specify code font if there is no description.
- *iStartSel* is the start of selection offset in the file. Specify null if not used.
- *iEndSel* is the end of selection offset in the file. Specify code font if not used.
- *colNdata* is a string to use for each column.

Returns

A Boolean value, `true` if the item was added successfully, `false` otherwise.

resWin.setCallbackCommands()

Availability

Dreamweaver 4.0

Description

Tells the Results window on which commands to call the `processFile()` method. If this function is not called, the command that created the Results window is called.

Arguments

arrCmdNames

arrCmdNames is an array of command names on which to call the `processFile()` method.

Returns

Nothing.

resWin.setColumnWidths()

Availability

Dreamweaver 4.0

Description

Sets the width of each column.

Arguments

arrWidth

arrWidth is an array of integers representing the widths to use for each column in the control.

Returns

Nothing.

resWin.setFileList()

Availability

Dreamweaver 4.0

Description

Gives the Results window a list of files, directories, or both to call a set of commands to process.

Arguments

arrFilePaths, *bRecursive*

- *arrFilePaths* is an array of file or folder paths to iterate through.
- *bRecursive* is a Boolean value indicating whether the iteration should be recursive (true) or not (false).

Returns

Nothing.

resWin.setTitle()

Availability

Dreamweaver 4.0

Description

Sets the title of the window.

Arguments

strTitle

strTitle is the new name of the floating panel.

Returns

Nothing.

resWin.startProcessing()

Availability

Dreamweaver 4.0

Description

Starts processing the file.

Arguments

None.

Returns

Nothing.

resWin.stopProcessing()

Availability

Dreamweaver 4.0

Description

Stops processing the file.

Arguments

None.

Returns

Nothing.

Selection functions

Selection functions get and set the selection in open documents. For information on getting or setting the selection in the Site window, see “Site functions” on page 348.

dom.getSelectedNode()

Availability

Dreamweaver 3.0

Description

Gets the selected node. This function is equivalent to calling `dom.getSelection()` and then passing the return value to `dom.offsetsToNode()`.

Arguments

None.

Returns

The tag, text, or comment object that completely contains the specified range of characters.

Enabler

None.

`dom.getSelection()`

Availability

Dreamweaver 3.0

Description

Gets the selection, expressed as character offsets into the document's HTML source code.

Arguments

bAllowMultiple

bAllowMultiple is a Boolean value indicating whether the function should return multiple offsets if more than one table cell, image map hotspot, or layer is selected. If this argument is omitted, it defaults to `false`.

Returns

For simple selections, an array containing two integers. The first integer is the character offset of the beginning of the selection. The second integer is the character offset of the end of the selection. If the two numbers are the same, then the current selection is an insertion point.

For complex selections (multiple table cells, multiple layers, or multiple image map hotspots), an array containing $2n$ integers, where n is the number of selected items. The first integer in each pair is the character offset of the beginning of the selection (including the opening TD, DIV, SPAN, LAYER, ILAYER, or MAP tag); the second integer in each pair is the character offset of the end of the selection (including the closing TD, DIV, SPAN, LAYER, ILAYER, or MAP tag). If multiple table rows are selected, the offsets of each cell in each row are returned. The selection never includes the TR tags.

Enabler

None.

dom.nodeToOffsets()

Availability

Dreamweaver 3.0

Description

Gets the position of a specific node in the DOM tree, expressed as character offsets into the document's HTML source code. Valid for any document on a local drive.

Arguments

node

node must be a tag, comment, or range of text that is a node in the tree returned by `dreamweaver.getDocumentDOM()`.

Returns

An array containing two integers. The first integer is the character offset of the beginning of the tag, text, or comment; the second integer is the character offset of the end of the node, from the beginning of the HTML document.

Enabler

None.

Example

The following code selects the first image object in the current document:

```
var theDOM = dw.getDocumentDOM();
var theImg = theDOM.images[0];
var offsets = theDom.nodeToOffsets(theImg);
theDom.setSelection(offsets[0], offsets[1]);
```

dom.offsetsToNode()

Availability

Dreamweaver 3.0

Description

Gets the object in the DOM tree that completely contains the range of characters between the specified begin and end points. Valid for any document on a local drive.

Arguments

offsetBegin, offsetEnd

The arguments are the begin and end points, respectively, of a range of characters, expressed as character offsets from the beginning of the document's HTML source code.

Returns

The tag, text, or comment object that completely contains the specified range of characters.

Enabler

None.

Example

The following code displays an alert if the selection is an image:

```
var offsets = dom.getSelection();
var theSelection = dreamweaver.offsetsToNode(offsets[0], ↵
offsets[1]);
if (theSelection.nodeType == Node.ELEMENT_NODE && ↵
theSelection.tagName == 'IMG'){
    alert('The current selection is an image.');
```

dom.selectAll()

Availability

Dreamweaver 3.0

Description

Performs a Select All operation.

Note: In most cases this function selects all content in the active document. In some cases (for example, when the insertion point is inside a table), however, it selects only part of the active document. To set the selection to the entire document, use `dom.setSelection()`.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.selectTable()

Availability

Dreamweaver 3.0

Description

Selects an entire table.

Arguments

None.

Returns

Nothing.

Enabler

`dom.canSelectTable()`

dom.setSelectedNode()

Availability

Dreamweaver 3.0

Description

Sets the selected node. This function is equivalent to calling `dom.nodeToOffsets()` and then passing the return value to `dom.setSelection()`.

Arguments

node, {*bSelectInside*}, {*bJumpToNode*}

- *node* is a text, comment, or element node in the document.
- *bSelectInside* is a Boolean value indicating whether to select the `innerHTML` of the node. This argument is relevant only if *node* is an element node, and it defaults to `false` if omitted.
- *bJumpToNode* is a Boolean value indicating whether to scroll the Document window, if necessary, to make the selection visible. If omitted, this argument defaults to `false`.

Returns

Nothing

Enabler

None.

dom.setSelection()

Availability

Dreamweaver 3.0

Description

Sets the selection in the document.

Arguments

offsetBegin, offsetEnd

The arguments are the begin and end points, respectively, for the new selection, expressed as character offsets into the document's HTML source code. If the two numbers are the same, the new selection is an insertion point. If the new selection is not a valid HTML selection, it is expanded to include the characters in the first valid HTML selection. For example, if *offsetBegin* and *offsetEnd* define the range SRC="myImage.gif" within , the selection expands to include the entire IMG tag.

Returns

Nothing.

Enabler

None.

dreamweaver.selectAll()

Availability

Dreamweaver 3.0

Description

Performs a Select All operation in the active Document window or the Site window; or, on the Macintosh, the edit field that has focus in a dialog box or floating panel.

Note: If the operation takes place in the active document, in most cases it selects all content in the active document. In some cases (for example, when the insertion point is inside a table), however, it selects only part of the active document. To set the selection to the entire document, use `dom.setSelection()`.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.canSelectAll()`

Site functions

Site functions handle operations related to files in the site files or site map. These functions create links between files; get, put, check in, and check out files; select and deselect files; create and remove files; get information about the sites that the user has defined; and more.

`dreamweaver.loadSitesFromPrefs()`

Availability

Dreamweaver 4.0

Description

Loads the site information for all sites from the system registry (Windows) or the Dreamweaver preferences file (Macintosh) into Dreamweaver. If a site is connected to a remote server when this function is called, the site will be automatically disconnected.

Arguments

None.

Returns

Nothing.

`dreamweaver.saveSitesToPrefs()`

Availability

Dreamweaver 4.0

Description

Saves all information for each site the user has defined to the system registry (Windows) or the Dreamweaver preferences file (Macintosh).

Arguments

None.

Returns

Nothing.

site.addLinkToExistingFile()

Availability

Dreamweaver 3.0

Description

Opens the Select HTML File dialog box to let the user select a file, and then creates a link from the selected document to that file.

Arguments

None.

Returns

Nothing.

Enabler

site.canAddLink()

site.addLinkToNewFile()

Availability

Dreamweaver 3.0

Description

Opens the Link to New File dialog box to let the user specify details for the new file, and then creates a link from the selected document to that file.

Arguments

None.

Returns

Nothing.

Enabler

site.canAddLink()

site.canEditColumns()

Description

Determines whether or not a site exists.

Arguments

None.

Returns

true if a site exists.

site.changeLinkSitewide()

Availability

Dreamweaver 3.0

Description

Opens the Change Link Sitewide dialog box.

Arguments

None.

Returns

Nothing.

Enabler

None.

site.changeLink()

Availability

Dreamweaver 3.0

Description

Opens the Select HTML File dialog box to let the user select a new file for the link.

Arguments

None.

Returns

Nothing.

Enabler

site.canChangeLink()

site.checkIn()

Availability

Dreamweaver 3.0

Description

Checks in the selected files and handles dependent files in one of the following ways:

- If the user selected Prompt on Put/Check In in the Site FTP preferences, the Dependent Files dialog box appears.
- If the user previously selected the Don't Show Me Again option in the Dependent Files dialog box and then clicked Yes, dependent files are uploaded and no dialog box appears.
- If the user previously selected the Don't Show Me Again option in the Dependent Files dialog box and then clicked No, dependent files are not uploaded and no dialog box appears.

Arguments

siteOrURL

siteOrURL must be the keyword "site", indicating that the function should act on the selection in the Site window or on the URL for a single file.

Returns

Nothing.

Enabler

site.canCheckIn()

site.checkLinks()

Availability

Dreamweaver 3.0

Description

Opens the Link Checker dialog box and checks links in the specified files.

Arguments

scopeOfCheck

scopeOfCheck specifies where links will be checked. It must be "document", "selection", or "site".

Returns

Nothing.

Enabler

None.

site.checkOut()

Availability

Dreamweaver 3.0

Description

Checks out the selected files and handles dependent files in one of the following ways:

- If the user selected Prompt on Get/Check Out in the Site FTP preferences, the Dependent Files dialog box appears.
- If the user previously selected the Don't Show Me Again option in the Dependent Files dialog box and then clicked Yes, dependent files are downloaded and no dialog box appears.
- If the user previously selected the Don't Show Me Again option in the Dependent Files dialog box and then clicked No, dependent files are not downloaded and no dialog box appears.

Arguments

siteOrURL

siteOrURL must be the keyword "site", indicating that the function should act on the selection in the Site window or on the URL for a single file.

Returns

Nothing.

Enabler

site.canCheckOut()

site.checkTargetBrowsers()

Availability

Dreamweaver 3.0

Description

Runs a target browser check on the selected files.

Arguments

None.

Returns

Nothing.

Enabler

None.

site.defineSites()

Availability

Dreamweaver 3.0

Description

Opens the Define Sites dialog box.

Arguments

None.

Returns

Nothing.

Enabler

None.

site.deleteSelection()

Availability

Dreamweaver 3.0

Description

Deletes the selected files.

Arguments

None.

Returns

Nothing.

Enabler

None.

site.editColumns()

Description

Displays the Define Sites dialog box with the file view columns section showing.

Arguments

None.

Returns

Nothing.

site.locateInSite()

Availability

Dreamweaver 3.0

Description

Locates the specified file or files in the specified pane of the Site window, and selects the found files.

Arguments

localOrRemote, *siteOrURL*

- *localOrRemote* must be either "local" or "remote".
- *siteOrURL* must be the keyword "site", indicating that the function should act on the selection in the Site window or on the URL for a single file.

Returns

Nothing.

Enabler

site.canLocateInSite()

site.findLinkSource()

Availability

Dreamweaver 3.0

Description

Opens the file that contains the selected link or dependent file, and highlights the text of the link or the reference to the dependent in that file. This function operates only on files in the Site Map view.

Arguments

None.

Returns

Nothing.

Enabler

site.canFindLinkSource()

site.get()

Availability

Dreamweaver 3.0

Description

Gets the specified files and handles dependent files in one of the following ways:

- If the user selected Prompt on Get/Check Out in the Site FTP preferences, the Dependent Files dialog box appears.
- If the user at some point selected the Don't Show Me Again option in the Dependent Files dialog box and then clicked Yes, dependent files are downloaded and no dialog box appears.
- If the user at some point selected the Don't Show Me Again option in the Dependent Files dialog box and then clicked No, dependent files are not downloaded and no dialog box appears.

Arguments

siteOrURL

siteOrURL must be the keyword "site", indicating that the function should act on the selection in the Site window or on the URL for a single file.

Returns

Nothing.

Enabler

site.canGet()

site.getCheckOutUser()

Availability

Dreamweaver 3.0

Description

Gets the login and check out name associated with the current site.

Arguments

None.

Returns

A string containing a login and check out name, if defined, or an empty string if check in/check out is disabled.

Enabler

None.

Example

A call to `site.getCheckOutUser()` might return "lori (loriLaptop)". If no check out name is specified, only the login is returned (for example, "lori").

site.getCheckOutUserForFile()

Availability

Dreamweaver 3.0

Description

Gets the login and check out name of the user that has the specified file checked out.

Arguments

fileName

fileName is the path to the file being inquired about, expressed as a file:// URL.

Returns

A string containing the login and check out name of the user that has the file checked out, or an empty string if the file is not checked out.

Enabler

None.

Example

A call to `site.getCheckOutUserForFile("file:///C:/sites/avocado8/index.html")` might return "lori (loriLaptop)". If no check out name is specified, only the login is returned (for example, "lori").

site.getConnectionState()

Availability

Dreamweaver 3.0

Description

Gets the current connection state.

Arguments

None.

Returns

A Boolean value indicating whether the remote site is connected.

Enabler

`site.canConnect()`

site.getCurrentSite()

Availability

Dreamweaver 3.0

Description

Gets the current site.

Arguments

None.

Returns

A string containing the name of the current site.

Enabler

None.

Example

If you defined several sites, a call to `site.getCurrentSite()` returns the one that is currently showing in the Current Sites List in the Site window.

site.getFocus()

Availability

Dreamweaver 3.0

Description

Determines which pane of the Site window is in focus.

Arguments

None.

Returns

One of the following strings: "local", "remote", or "site map".

Enabler

None.

site.getLinkVisibility()

Availability

Dreamweaver 3.0

Description

Checks whether all of the selected links in the site map are visible (that is, not marked hidden).

Arguments

None.

Returns

A Boolean value indicating whether all of the selected links are visible.

Enabler

None.

site.getSelection()

Availability

Dreamweaver 3.0

Description

Determines which files are currently selected in the Site window.

Arguments

None.

Returns

An array of strings representing the paths of the selected files and folders, expressed as file:// URLs, or an empty array if no files or folders are selected.

Enabler

None.

site.getSites()

Availability

Dreamweaver 3.0

Description

Gets a list of the defined sites.

Arguments

None.

Returns

An array of strings representing the names of the defined sites, or an empty array if no sites are defined.

Enabler

None.

site.invertSelection()

Availability

Dreamweaver 3.0

Description

Inverts the selection in the site map.

Arguments

None.

Returns

Nothing.

Enabler

None.

site.makeEditable()

Availability

Dreamweaver 3.0

Description

Turns off the read-only flag on the selected files.

Arguments

None.

Returns

Nothing.

Enabler

site.canMakeEditable()

site.makeNewDreamweaverFile()

Availability

Dreamweaver 3.0

Description

Creates a new Dreamweaver file in the Site window (in the same directory as the first selected file or folder).

Arguments

None.

Returns

Nothing.

Enabler

site.canMakeNewFileOrFolder()

site.makeNewFolder()

Availability

Dreamweaver 3.0

Description

Creates a new folder in the Site window (in the same directory as the first selected file or folder).

Arguments

None.

Returns

Nothing.

Enabler

site.canMakeNewFileOrFolder()

site.newHomePage()

Availability

Dreamweaver 3.0

Description

Opens the New Home Page dialog box to let the user create a new home page.

Note: This function operates only on files in the Site Map view.

Arguments

None.

Returns

Nothing.

Enabler

None.

site.newSite()

Availability

Dreamweaver 3.0

Description

Opens the Site Definition Dialog box for a new, unnamed site.

Arguments

None.

Returns

Nothing.

Enabler

None.

site.open()

Availability

Dreamweaver 3.0

Description

Opens the files that are currently selected in the Site window. If any folders are selected, they are expanded in the Site Files view.

Arguments

None.

Returns

Nothing.

Enabler

site.canOpen()

site.put()

Availability

Dreamweaver 3.0

Description

Puts the selected files and handles dependent files in one of the following ways:

- If the user selected Prompt on Put/Check In in the Site FTP preferences, the Dependent Files dialog box appears.
- If the user previously selected the Don't Show Me Again option in the Dependent Files dialog box and then clicked Yes, dependent files are uploaded and no dialog box appears.
- If the user previously selected the Don't Show Me Again option in the Dependent Files dialog box and then clicked No, dependent files are not uploaded and no dialog box appears.

Arguments

siteOrURL

siteOrURL must be the keyword "site", indicating that the function should act on the selection in the Site window or on the URL for a single file.

Returns

Nothing.

Enabler

site.canPut()

site.recreateCache()

Availability

Dreamweaver 3.0

Description

Recreates the cache for the current site.

Arguments

None.

Returns

Nothing.

Enabler

site.canRecreateCache()

site.refresh()

Availability

Dreamweaver 3.0

Description

Refreshes the file listing on the specified side of the Site window.

Arguments

whichSide

whichSide must be "local", or "remote". If the site map has focus and *whichSide* is "local", the site map refreshes.

Returns

Nothing.

Enabler

site.canRefresh()

site.remotelsValid()

Availability

Dreamweaver 3.0

Description

Determines whether the remote site is valid.

Arguments

None.

Returns

A Boolean value indicating whether a remote site has been defined and, if the server type is Local/Network, whether the drive is mounted.

Enabler

None.

site.removeLink()

Availability

Dreamweaver 3.0

Description

Removes the selected link from the document above it in the site map.

Arguments

None.

Returns

Nothing.

Enabler

site.canRemoveLink()

site.renameSelection()

Availability

Dreamweaver 3.0

Description

Turns the name of the selected file into an edit field, allowing the user to rename the file. If more than one file is selected, this function acts on the last selected file.

Arguments

None.

Returns

Nothing.

Enabler

None.

site.saveAsImage()

Availability

Dreamweaver 3.0

Description

Opens the Save As dialog box to let the user save the site map as an image.

Arguments

fileType

fileType is the type of image that should be saved. Valid values for Windows are "bmp" and "png"; valid values for Macintosh are "pict" and "jpeg". If the argument is omitted, or if the value is not valid on the current platform, the default is "bmp" in Windows and "pict" on the Macintosh.

Returns

Nothing.

Enabler

None.

site.selectAll()

Availability

Dreamweaver 3.0

Description

Selects all files in the active view (either the site map or the site files).

Arguments

None.

Returns

Nothing.

Enabler

None.

site.selectHomePage()

Availability

Dreamweaver 3.0

Description

Opens the Open File dialog box to let the user choose a new home page.

Note: This function operates only on files in the Site Map view.

Arguments

None.

Returns

Nothing.

Enabler

None.

site.selectNewer()

Availability

Dreamweaver 3.0

Description

Selects all files that are newer on the specified side of the Site window.

Arguments

whichSide

whichSide must be either "local" or "remote".

Returns

Nothing.

Enabler

site.canSelectNewer()

site.setAsHomePage()

Availability

Dreamweaver 3.0

Description

Designates the file that is selected in the Site Files view as the home page for the site.

Arguments

None.

Returns

Nothing.

Enabler

None.

site.setConnectionState()

Availability

Dreamweaver 3.0

Description

Sets the connection state of the current site.

Arguments

bConnected

bConnected is a Boolean value indicating a connection (true) or not (false) to the current site.

Returns

Nothing.

Enabler

None.

site.setCurrentSite()

Availability

Dreamweaver 3.0

Description

Opens the specified site in the local pane of the Site window.

Arguments

whichSite

whichSite is the name of a defined site (as it appears in the Current Sites list in the Site window or the Define Sites dialog box).

Returns

Nothing.

Enabler

None.

Example

If three sites are defined (for example, avocado8, dreamcentral, and testsite), a call to `site.setCurrentSite("dreamcentral");` makes dreamcentral the current site.

site.setFocus()

Availability

Dreamweaver 3.0

Description

Gives focus to the specified pane of the Site window. If the specified pane is not showing, displays the pane and gives it focus.

Arguments

whichPane

whichPane must be one of the following strings: "local", "remote", or "site map".

Returns

Nothing.

Enabler

None.

site.setLayout()

Availability

Dreamweaver 3.0

Description

Opens the Site Map Layout pane of the Site Definition dialog box.

Arguments

None.

Returns

Nothing.

Enabler

`site.canSetLayout()`

site.setLinkVisibility()

Availability

Dreamweaver 3.0

Description

Shows or hides the current link.

Arguments

bShow

bShow is a Boolean value indicating whether to remove the Hidden designation from the current link.

Returns

Nothing.

Enabler

None.

site.setSelection()

Availability

Dreamweaver 3.0

Description

Selects files or folders in the active pane in the Site window.

Arguments

arrayOfURLs

arrayOfURLs is an array of strings, each a path to a file or folder in the current site, expressed as a file:// URL.

Note: Leave off the trailing slash (/) when specifying folder paths.

Returns

Nothing.

Enabler

None.

site.synchronize()

Availability

Dreamweaver 3.0

Description

Opens the Synchronize Files dialog box.

Arguments

None.

Returns

Nothing.

Enabler

site.canSynchronize()

site.undoCheckOut()

Availability

Dreamweaver 3.0

Description

Removes the lock files that are associated with the specified files from the local and remote sites, and replaces the local copy of the specified files with the remote copy.

Arguments

siteorURL

siteorURL must be the keyword "site", indicating that the function should act on the selection in the Site window or on the URL for a single file.

Returns

Nothing.

Enabler

site.canUndoCheckOut()

site.viewAsRoot()

Availability

Dreamweaver 3.0

Description

Temporarily moves the selected file to the top position in the site map.

Arguments

None.

Returns

Nothing.

Enabler

`site.canViewAsRoot()`

Source View functions

Source View functions include operations related to text editing document source code (and any subsequent impact on the Design view). The functions in this section enable you to add navigational controls to source views within a split document view, the Code inspector, and the JavaScript Debugger window.

dom.isDesignViewUpdated()

Availability

Dreamweaver 4.0

Description

Determines whether or not the Design view and Text view content is synchronized for those Dreamweaver operations that require a valid document state.

Arguments

None.

Returns

`true` if the Design view (WYSIWIG) is synchronized with the text in the Text view, `false` otherwise.

dom.isSelectionValid()

Availability

Dreamweaver 4.0

Description

Determines whether or not a selection is valid, or if it needs to be moved before an operation occurs.

Arguments

None.

Returns

true if the current selection is in a valid piece of code. If the document has not been synchronized, returns false (because the selection is not updated).

dom.source.arrowDown()

Availability

Dreamweaver 4.0

Description

Moves the insertion point down the source view document, line by line. If content is already selected, this function extends the selection line by line.

Arguments

{nTimes}, {bShiftIsDown}

- *nTimes* is the number of lines the insertion point will move. If *nTimes* is omitted, the default is 1.
- *bShiftIsDown* is a Boolean indicating whether or not content is being selected. If *bShiftIsDown* is true, the content is selected.

Returns

Nothing.

`dom.source.arrowLeft()`

Availability

Dreamweaver 4.0

Description

Moves the insertion point to the left in the current line of the Source view. If content is already selected, this function extends the selection to the left.

Arguments

{nTimes}, {bShiftIsDown}

- *nTimes* is the number of characters the insertion point will move. If *nTimes* is omitted, the default is 1.
- *bShiftIsDown* is a Boolean indicating whether or not content is being selected. If *bShiftIsDown* is true, the content is selected.

Returns

Nothing.

`dom.source.arrowRight()`

Availability

Dreamweaver 4.0

Description

Moves the insertion point to the right in the current line of the Source view. If content is already selected, this function extends the selection to the right.

Arguments

{nTimes}, {bShiftIsDown}

- *nTimes* is the number of characters the insertion point will move. If *nTimes* is omitted, the default is 1.
- *bShiftIsDown* is a Boolean indicating whether or not content is being selected. If *bShiftIsDown* is true, the content is selected.

Returns

Nothing.

`dom.source.arrowUp()`

Availability

Dreamweaver 4.0

Description

Moves the insertion point up the source view document, line by line. If content is already selected, this function extends the selection line by line.

Arguments

{nTimes}, *{bShiftIsDown}*

- *nTimes* is the number of lines the insertion point will move. If *nTimes* is omitted, the default is 1.
- *bShiftIsDown* is a Boolean indicating whether or not content is being selected. If *bShiftIsDown* is true, the content is selected.

Returns

Nothing.

`dom.source.balanceBracesTextView()`

Availability

Dreamweaver 4.0

Description

Source view extension that enables parentheses balancing. You can call `dom.source.balanceBracesTextView()` to extend a currently highlighted selection or insertion point from the start of the surrounding parenthetical statement to the end of the statement to balance the following characters: `[]`, `{ }` and `()`. Subsequent calls will expand the selection through further levels of punctuation nesting.

Arguments

None.

Returns

Nothing.

`dom.source.endOfDocument()`

Availability

Dreamweaver 4.0

Description

Places the insertion point at the end of the current source view document. If content is already selected, this function extends the selection to the end of the document.

Arguments

bShiftIsDown

A Boolean indicating whether or not content is being selected. If *bShiftIsDown* is `true`, the content is selected.

Returns

Nothing.

`dom.source.endOfLine()`

Availability

Dreamweaver 4.0

Description

Places the insertion point at the end of the current line. If content is already selected, this function extends the selection to the end of the current line.

Arguments

bShiftIsDown

A Boolean indicating whether or not content is being selected. If *bShiftIsDown* is `true`, the content is selected.

Returns

Nothing.

dom.source.endPage()

Availability

Dreamweaver 4.0

Description

Moves the insertion point to the end of the current page or to the end of the next page (if the insertion point is already at the end of a page). If content is already selected, this function extends the selection page by page.

Arguments

{nTimes}, {bShiftIsDown}

- *nTimes* is the number of pages the insertion point will move. If *nTimes* is omitted, the default is 1.
- *bShiftIsDown* is a Boolean indicating whether or not content is being selected. If *bShiftIsDown* is true, the content is selected.

Returns

Nothing.

dom.source.getCurrentLines()

Availability

Dreamweaver 4.0

Description

Returns the line numbers for the specified offset locations from the beginning of the document.

Arguments

startOffset, endOffset

- *startOffset* is an integer representing the beginning line of text.
- *endOffset* is an integer representing the ending line of text.

Returns

An array of two numbers representing the beginning and ending lines of the text between (and including) *startOffset* and *endOffset*. If *startOffset* and *endOffset* are not valid, returns -1.

`dom.source.getSelection()`

Description

Gets the selection in the current document, expressed as character offsets into the document's Code view.

Arguments

None.

Returns

A pair of integers representing offsets from the beginning of the source document. The first integer is the beginning of the selection, and the second is the end of the selection. If the two numbers are equal, the selection is an IP. If there is no selection in the source, both numbers are -1.

`dom.source.getText()`

Availability

Dreamweaver 4.0

Description

Returns the text string in the source between the designated offsets.

Arguments

start, end

- *start* is an integer representing the offset from the beginning of the document.
- *end* is an integer representing the end of the document.

Returns

A string representing the text in the HTML source code between the offsets *start* and *end*.

`dom.source.indentTextView()`

Availability

Dreamweaver 4.0

Description

Moves selected source view text one tab stop to the right.

Arguments

None

Returns

Nothing.

`dom.source.insert()`

Description

Inserts the specified string into the HTML source code at the specified offset from the beginning of the source file. If the offset is not greater than, or equal to, zero, the insertion fails and the function returns `false`.

Arguments

offset, *string*

- *offset* is the offset from the beginning of the file where the string should be inserted.
- *string* is the string to insert.

Returns

`true` if successful, `false` if not.

`dom.source.nextWord()`

Availability

Dreamweaver 4.0

Description

Moves the insertion point to the beginning of the next word (or words, if specified) in the Source view. If content is already selected, this function extends the selection to the right.

Arguments

{nTimes}, *{bShiftIsDown}*

- *nTimes* is the number of words the insertion point will move. If *nTimes* is omitted, the default is 1.
- *bShiftIsDown* is a Boolean indicating whether or not content is being selected. If *bShiftIsDown* is `true`, the content is selected.

Returns

Nothing.

`dom.source.outdentTextView()`

Availability

Dreamweaver 4.0

Description

Moves selected source view text one tab stop to the left.

Arguments

None

Returns

Nothing.

`dom.source.pageDown()`

Availability

Dreamweaver 4.0

Description

Moves the insertion point down the source view document, page by page. If content is already selected, this function extends the selection page by page.

Arguments

{nTimes}, {bShiftIsDown}

- *nTimes* is the number of pages the insertion point will move. If *nTimes* is omitted, the default is 1.
- *bShiftIsDown* is a Boolean indicating whether or not content is being selected. If *bShiftIsDown* is true, the content is selected.

Returns

Nothing.

`dom.source.pageUp()`

Availability

Dreamweaver 4.0

Description

Moves the insertion point up the source view document, page by page. If content is already selected, this function extends the selection page by page.

Arguments

{nTimes}, *{bShiftIsDown}*

- *nTimes* is the number of pages the insertion point will move. If *nTimes* is omitted, the default is 1.
- *bShiftIsDown* is a Boolean indicating whether or not content is being selected. If *bShiftIsDown* is true, the content is selected.

Returns

Nothing.

`dom.source.previousWord()`

Availability

Dreamweaver 4.0

Description

Moves the insertion point to the beginning of the previous word (or words if specified) in the source view. If content is already selected, this function extends the selection to the left.

Arguments

{nTimes}, *{bShiftIsDown}*

- *nTimes* is the number of words the insertion point will move. If *nTimes* is omitted, the default is 1.
- *bShiftIsDown* is a Boolean indicating whether or not content is being selected. If *bShiftIsDown* is true, the content is selected.

Returns

Nothing.

`dom.source.replaceRange()`

Availability

Dreamweaver 4.0

Description

Replaces the range of source text between *startOffset* and *endOffset* with *string*. If *startOffset* is greater than *endOffset* or if either offset is not a positive integer, does nothing and returns `false`. If *endOffset* is greater than the number of characters in the file, replaces the range between *startOffset* and the end of the file. If both *startOffset* and *endOffset* are greater than the number of characters in the file, inserts the text at the end of the file.

Arguments

startOffset, *endOffset*, *string*

- *startOffset* is the offset indicating the beginning of the block to replace.
- *endOffset* is the offset indicating the end of the block to replace.
- *string* is the string to insert.

Returns

`true` if successful, `false` if not.

`dom.source.scrollToEndFile()`

Availability

Dreamweaver 4.0

Description

Scrolls the Source view to the bottom of the document file without moving the insertion point.

Arguments

None.

Returns

Nothing.

`dom.source.scrollLineDown()`

Availability

Dreamweaver 4.0

Description

Scrolls the Source view down line by line without moving the insertion point.

Arguments

nTimes

nTimes is the number of lines to scroll. If *nTimes* is omitted, the default is 1.

Returns

Nothing.

`dom.source.scrollLineUp()`

Availability

Dreamweaver 4.0

Description

Scrolls the Source view up line by line without moving the insertion point.

Arguments

nTimes

nTimes is the number of lines to scroll. If *nTimes* is omitted, the default is 1.

Returns

Nothing.

`dom.source.scrollPageDown()`

Availability

Dreamweaver 4.0

Description

Scrolls the Source view down page by page without moving the insertion point.

Arguments

nTimes

nTimes is the number of pages to scroll. If *nTimes* is omitted, the default is 1.

Returns

Nothing.

`dom.source.scrollPageUp()`

Availability

Dreamweaver 4.0

Description

Scrolls the Source view up page by page without moving the insertion point.

Arguments

nTimes

nTimes is the number of pages to scroll. If *nTimes* is omitted, the default is 1.

Returns

Nothing.

`dom.source.scrollTopFile()`

Availability

Dreamweaver 4.0

Description

Scrolls the Source view to the top of the document file without moving the insertion point.

Arguments

None.

Returns

Nothing.

`dom.source.selectParentTag()`

Availability

Dreamweaver 4.0

Description

Source view extension that enables tag balancing. You can call `dom.source.selectParentTag()` to extend a currently highlighted selection or insertion point from the surrounding open tag to the closing tag. Subsequent calls extend the selection to additional surrounding tags until there are no more enclosing tags.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dom.source.setCurrentLine()`

Availability

Dreamweaver 4.0

Description

Puts the insertion point at the beginning of the specified line. If *lineNumber* is not a positive integer, does nothing and returns `false`. Puts the insertion point at the beginning of the last line if *lineNumber* is larger than the number of lines in the source.

Arguments

lineNumber

lineNumber is the line at the beginning of which the insertion point is placed.

Returns

`true` if successful, `false` if not.

`dom.source.startOfDocument()`

Availability

Dreamweaver 4.0

Description

Places the insertion point at the beginning of the source view document. If content is already selected, this function extends the selection to the beginning of the document.

Arguments

bShiftIsDown

A Boolean indicating whether or not content is being selected. If *bShiftIsDown* is `true`, the content is selected.

Returns

Nothing.

dom.source.startOfLine()

Availability

Dreamweaver 4.0

Description

Places the insertion point at the beginning of the current line. If content is already selected, this function extends the selection to the beginning of the current line.

Arguments

bShiftIsDown

A Boolean indicating whether or not content is being selected. If *bShiftIsDown* is *true*, the content is selected.

Returns

Nothing.

dom.source.topPage()

Availability

Dreamweaver 4.0

Description

Moves the insertion point to the top of the current page or to the top of the previous page (if the insertion point is already at the top of a page). If content is already selected, this function extends the selection page by page.

Arguments

{nTimes}, *{bShiftIsDown}*

- *nTimes* is the number of pages the insertion point will move. If *nTimes* is omitted, the default is 1.
- *bShiftIsDown* is a Boolean indicating whether or not content is being selected. If *bShiftIsDown* is *true*, the content is selected.

Returns

Nothing.

`dom.source.wrapSelection()`

Availability

Dreamweaver 4.0

Description

Inserts the text of *startTag* before the current selection and the text of *endTag* after the current selection. The function then selects the entire range that has just been inserted. If the current selection was an insertion point, then the function places the insertion point between the *startTag* and *endTag*. *startTag* and *endTag* don't have to be tags; they can be any arbitrary text.

Arguments

startTag, *endTag*

- *startTag* is the text to insert at the beginning of the selection.
- *endTag* is the text to insert at the end of the selection.

Returns

Nothing.

`dom.synchronizeDocument()`

Availability

Dreamweaver 4.0

Description

Synchronizes the Design and Source views.

Arguments

None.

Returns

Nothing.

String manipulation functions

String manipulation functions help you get information about a string, as well as convert a string from Latin 1 encoding to platform-native encoding and back.

`dreamweaver.doURLEncoding()`

Availability

Dreamweaver 1.0

Description

This function takes a string and returns a URL-encoded string by replacing all spaces and special characters with specified entities.

Arguments

stringToConvert

Returns

A URL-encoded string.

Enabler

None.

Example

If the `URL.value` is "My URL-encoded string":

```
var URL = dw.doURLEncoding(theURL.value);  
returns "My%20URL-encoded%20string"
```


`dreamweaver.getTokens()`

Availability

Dreamweaver 1.0

Description

Accepts a string and splits it into tokens.

Arguments

searchString, *separatorCharacters*

- *searchString* is the string to be separated into tokens.
- *separatorCharacters* is the character or characters that signifies the end of a token. Separator characters in quoted strings are ignored. If *separatorCharacters* contains a space, all white-space characters (for example, tabs) are treated as separator characters as if you had explicitly specified them. Two or more consecutive white space characters are treated as a single separator.

Returns

An array of token strings.

Enabler

None.

Example

`dreamweaver.getTokens('foo("my arg1", 34)', '(),')` returns the tokens:

- `foo`
- `"my arg 1"`
- `34`

`dreamweaver.latin1ToNative()`

Availability

Dreamweaver 2.0

Description

Converts a string in Latin 1 encoding to the native encoding on the user's machine. This function is intended for displaying the user interface of an extension file in another language.

Note: This function has no effect in Windows because Windows encodings are already based on Latin 1.

Arguments

stringToConvert

stringToConvert is the (already translated) string to be converted from Latin 1 encoding to native encoding.

Returns

The converted string.

Enabler

None.

`dreamweaver.nativeToLatin1()`

Availability

Dreamweaver 2.0

Description

Converts a string in native encoding to Latin 1 encoding.

Note: This function has no effect in Windows because Windows encodings are already based on Latin 1.

Arguments

stringToConvert

stringToConvert is the string to be converted from native encoding to Latin 1 encoding.

Returns

The converted string.

Enabler

None.

The `dreamweaver.scanSourceString()` function is a utility function that analyzes a string and deciphers where the HTML tags, attributes, and directives are located. Here is a scenario for using the `dreamweaver.scanSourceString()` function:

- 1 You create an implementation for one or more of the seven callback functions.
- 2 You write a script that calls the `dreamweaver.scanSourceString()` function.
- 3 The `dreamweaver.scanSourceString()` function is passed a string containing HTML and pointers to the callback functions that you have written. For example, suppose the string of HTML is "hello".
- 4 Dreamweaver analyzes the string and determines that the string contains a font tag. Dreamweaver calls the callback functions in the following sequence:
 - the `openTagBegin()` function
 - the `attribute()` function (for the size attribute)
 - the `openTagEnd()` function
 - the `text()` function (for the "hello" string)
 - the `closeTagBegin()` and `closeTagEnd()` functions

`dreamweaver.scanSourceString` callback functions

As mentioned, there are seven callback functions that Dreamweaver calls:

- 1 Dreamweaver calls `openTagBegin()` for each opening tag (for example, ``, as opposed to ``) and each empty tag (for example, `` or `<hr>`). The `openTagBegin()` function accepts two arguments: the name of the tag (for example, "font" or "img") and the document offset, which is the number of bytes in the document before the beginning of the tag. The function returns `true` if scanning should continue, or `false` if it should stop.
- 2 After `openTagBegin()` executes, Dreamweaver calls `attribute()` for each HTML attribute. The `attribute()` function accepts two arguments: a string containing the attribute name (for example, "color" or "src") and a string containing the attribute value (for example, "#000000" or "foo.gif"). The `attribute()` function returns a Boolean indicating whether or not scanning should continue.
- 3 After all of the attributes in the tag have been scanned, Dreamweaver calls `openTagEnd()`. The `openTagEnd()` function accepts one argument: the document offset, which is the number of bytes in the document before the end of the opening tag. It returns a Boolean indicating whether or not scanning should continue.
- 4 Dreamweaver calls `closeTagBegin()` for each closing tag (for example, ``). The function accepts two arguments: the name of the tag to close (for example, "font") and the document offset, which is the number of bytes in the document before the beginning of the close tag. The function returns a Boolean indicating whether or not scanning should continue.
- 5 After `closeTagBegin()` returns, Dreamweaver calls the `closeTagEnd()` function. The `closeTagEnd()` function accepts one argument: the document offset, which is the number of bytes in the document before the end of the closing tag. It returns a Boolean indicating whether or not scanning should continue.
- 6 Dreamweaver calls the `directive()` function for each HTML comment, ASP script, JSP script, or PHP script. The `directive()` function accepts two arguments: a string containing the directive and the document offset, which is the number of bytes in the document before the end of the closing tag. The function returns a Boolean indicating whether or not scanning should continue.
- 7 Dreamweaver calls the `text()` function for each span of text in the document, that is, everything that is not a tag or a directive. Text spans include text that is not visible to the user, such as the text inside a `<title>` or `<option>` tag. The `text()` function accepts two arguments: a string containing the text and the document offset, which is the number of bytes in the document before the end of the closing tag. The `text()` function returns a Boolean indicating whether or not scanning should continue.

`dreamweaver.scanSourceString()`

Availability

Dreamweaver UltraDev 1.0

Description

Scans a string of HTML and finds the tags, attributes, directives, and text. For each tag, attribute, directive, and text span found, `scanSourceString()` invokes a callback function supplied by the caller. Dreamweaver supports the following callback functions: `openTagBegin()`, `openTagEnd()`, `closeTagBegin()`, `closeTagEnd()`, `directive()`, `attribute()`, and `text()`.

Arguments

HTMLstr, *parserCallbackObj*

- *HTMLstr* is a string containing HTML code.
- *parserCallbackObj* is a JavaScript object that has one or more of the following methods: `openTagBegin()`, `openTagEnd()`, `closeTagBegin()`, `closeTagEnd()`, `directive()`, `attribute()`, and `text()`. For best performance, *parserCallbackObj* should be a shared library defined using the C-Level Extensibility interface. Performance is also improved if *parserCallbackObj* defines only the callback functions that it needs.

Returns

A Boolean value indicating whether or not the operation completed successfully.

Table editing functions

Table functions add and remove table rows and columns, change column widths and row heights, convert measurements from pixels to percents and back, and perform other standard table editing tasks.

`dom.convertWidthsToPercent()`

Availability

Dreamweaver 3.0

Description

Converts all `WIDTH` attributes in the current table from pixels to percentages.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.convertWidthsToPixels()

Availability

Dreamweaver 4.0

Description

Converts all WIDTH attributes in the current table from percentages to pixels.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.decreaseColspan()

Availability

Dreamweaver 3.0

Description

Decreases the column span by 1.

Arguments

None.

Returns

Nothing.

Enabler

dom.canDecreaseColspan()

dom.decreaseRowspan()

Availability

Dreamweaver 3.0

Description

Decreases the row span by 1.

Arguments

None.

Returns

Nothing.

Enabler

dom.canDecreaseRowspan()

dom.deleteTableColumn()

Availability

Dreamweaver 3.0

Description

Removes the selected table column or columns.

Arguments

None.

Returns

Nothing.

Enabler

dom.canDeleteTableColumn()

dom.deleteTableRow()

Availability

Dreamweaver 3.0

Description

Removes the selected table row or rows.

Arguments

None.

Returns

Nothing.

Enabler

dom.canDeleteTableRow()

dom.doDeferredTableUpdate()

Availability

Dreamweaver 3.0

Description

If the Faster Table Editing option is selected in the General preferences, forces the table layout to reflect recent changes without moving the selection outside the table. This function has no effect if the Faster Table Editing option is not selected.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.getTableExtent()

Availability

Dreamweaver 3.0

Description

Gets the number of columns and rows in the selected table.

Arguments

None.

Returns

An array containing two whole numbers. The first array item is the number of columns, and the second array item is the number of rows. If no table was selected, nothing is returned.

Enabler

None.

dom.increaseColspan()

Availability

Dreamweaver 3.0

Description

Increases the column span by 1.

Arguments

None.

Returns

Nothing.

Enabler

dom.canIncreaseColspan()

dom.increaseRowspan()

Availability

Dreamweaver 3.0

Description

Increases the row span by 1.

Arguments

None.

Returns

Nothing.

Enabler

dom.canIncreaseRowspan()

`dom.insertTableColumns()`

Availability

Dreamweaver 3.0

Description

Inserts the specified number of table columns into the current table.

Arguments

numberOfCols, *bBeforeSelection*

- *numberOfCols* is the number of columns to insert.
- *bBeforeSelection* is a Boolean value indicating whether the columns should be inserted before the column that contains the selection.

Returns

Nothing.

Enabler

`dom.canInsertTableColumns()`

`dom.insertTableRows()`

Availability

Dreamweaver 3.0

Description

Inserts the specified number of table rows into the current table.

Arguments

numberOfRows, *bBeforeSelection*

- *numberOfRows* is the number of rows to insert.
- *bBeforeSelection* is a Boolean value indicating whether the rows should be inserted above the row that contains the selection.

Returns

Nothing.

Enabler

`dom.canInsertTableRows()`

dom.mergeTableCells()

Availability

Dreamweaver 3.0

Description

Merges the selected table cells.

Arguments

None.

Returns

Nothing.

Enabler

dom.canMergeTableCells()

dom.removeAllTableHeights()

Availability

Dreamweaver 3.0

Description

Removes all HEIGHT attributes from the selected table.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.removeAllTableWidths()

Availability

Dreamweaver 3.0

Description

Removes all WIDTH attributes from the selected table.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.setTableCellTag()

Availability

Dreamweaver 3.0

Description

Specifies the tag for the selected cell.

Arguments

tdOrTh

tdOrTh must be either "td" or "th".

Returns

Nothing.

Enabler

None.

dom.setTableColumns()

Availability

Dreamweaver 3.0

Description

Sets the number of columns in the selected table.

Arguments

numberOfCols

Returns

Nothing.

Enabler

None.

dom.setTableRows()

Availability

Dreamweaver 3.0

Description

Sets the number of rows in the selected table.

Arguments

numberOfRows

Returns

Nothing.

Enabler

None.

dom.showInsertTableRowsOrColumnsDialog()

Availability

Dreamweaver 3.0

Description

Opens the Insert Rows or Columns dialog box.

Arguments

None.

Returns

Nothing.

Enabler

dom.canInsertTableColumns() or dom.canInsertTableRows()

dom.splitTableCell()

Availability

Dreamweaver 3.0

Description

Splits the current table cell into the specified number of rows or columns. If one or both of the arguments is omitted, the Split Cells dialog box appears.

Arguments

{colsOrRows}, *{numberToSplitInto}*

- *colsOrRows*, if supplied, must be either "columns" or "rows".
- *numberToSplitInto*, if supplied, is the number of rows or columns to split the cell into.

Returns

Nothing.

Enabler

dom.canSplitTableCell()

Timeline functions

Timeline functions act on timelines. They add, remove, and change objects in a timeline; add behaviors, frames, and keyframes to a timeline; specify whether the timeline should play and loop automatically; and more. All of the functions in this section are methods of the `dreamweaver.timelineInspector` object because they affect the contents of the Timelines panel.

dreamweaver.timelineInspector.addBehavior()

Availability

Dreamweaver 3.0

Description

Opens the Behaviors panel and automatically supplies the correct *onFrameN* event (where *N* is the frame marked by the playback head) when the user chooses an action and clicks OK.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.timelineInspector.addFrame()`

Availability

Dreamweaver 3.0

Description

Adds a frame to the current timeline at the frame that contains the playback head.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.timelineInspector.canAddFrame()`

`dreamweaver.timelineInspector.addKeyframe()`

Availability

Dreamweaver 3.0

Description

Adds a keyframe to the selected animation bar at the frame that contains the playback head.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.timelineInspector.canAddKeyFrame()`

`dreamweaver.timelineInspector.addObject()`

Availability

Dreamweaver 3.0

Description

Adds the currently selected object to the timeline.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.timelineInspector.addTimeline()`

Availability

Dreamweaver 3.0

Description

Adds a new timeline to the current document.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.timelineInspector.changeObject()`

Availability

Dreamweaver 3.0

Description

Opens the Change Object dialog box.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.timelineInspector.canChangeObject()`

`dreamweaver.timelineInspector.getAutoplay()`

Availability

Dreamweaver 3.0

Description

Gets the state of the Autoplay option for the current timeline.

Arguments

None.

Returns

A Boolean value indicating whether the Autoplay option is selected.

Enabler

None.

`dreamweaver.timelineInspector.getCurrentFrame()`

Availability

Dreamweaver 3.0

Description

Gets the current frame of the current timeline.

Arguments

None.

Returns

A frame number.

Enabler

None.

`dreamweaver.timelineInspector.getLoop()`

Availability

Dreamweaver 3.0

Description

Gets the state of the Loop option for the current timeline.

Arguments

None.

Returns

A Boolean value indicating whether the Loop option is selected.

Enabler

None.

`dreamweaver.timelineInspector.recordPathOfLayer()`

Availability

Dreamweaver 3.0

Description

Records the path of a layer as the user drags it.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.timelineInspector.removeBehavior()`

Availability

Dreamweaver 3.0

Description

Removes the selected behavior from the timeline.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.timelineInspector.canRemoveBehavior()`

`dreamweaver.timelineInspector.removeFrame()`

Availability

Dreamweaver 3.0

Description

Removes the selected frame from the timeline.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.timelineInspector.canRemoveFrame()`

`dreamweaver.timelineInspector.removeKeyframe()`

Availability

Dreamweaver 3.0

Description

Removes the selected keyframe from an animation bar.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.timelineInspector.canRemoveKeyFrame()`

`dreamweaver.timelineInspector.removeObject()`

Availability

Dreamweaver 3.0

Description

Removes the currently selected object from the timeline.

Arguments

None.

Returns

Nothing.

Enabler

`dreamweaver.timelineInspector.canRemoveObject()`

`dreamweaver.timelineInspector.removeTimeline()`

Availability

Dreamweaver 3.0

Description

Removes the current timeline from the document.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.timelineInspector.renameTimeline()`

Availability

Dreamweaver 3.0

Description

Opens the Rename Timeline dialog box for the current timeline.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.timelineInspector.setAutoplay()`

Availability

Dreamweaver 3.0

Description

Sets the Autoplay option for the current timeline.

Arguments

bAutoplay

bAutoplay is a Boolean value indicating whether to turn on the Autoplay option.

Returns

Nothing.

Enabler

None.

`dreamweaver.timelineInspector.setCurrentFrame()`

Availability

Dreamweaver 3.0

Description

Moves the playback head to the specified frame.

Arguments

frameNumber

Returns

Nothing.

Enabler

None.

`dreamweaver.timelineInspector.setLoop()`

Availability

Dreamweaver 3.0

Description

Sets the Loop option for the current timeline.

Arguments

bLoop

bLoop is a Boolean value indicating whether to turn on the Loop option.

Returns

Nothing.

Enabler

None.

Toggle functions

Toggle functions get and set various options either on or off.

`dom.getEditNoFramesContent()`

Availability

Dreamweaver 3.0

Description

Gets the current state of the Modify > Frameset > Edit NoFrames Content option.

Arguments

None.

Returns

A Boolean value indicating whether the NOFRAMES content is the active view (true) or not (false).

Enabler

None.

dom.getHideAllVisualAids()

Availability

Dreamweaver 4.0

Description

Determines whether or not visual aids are set as hidden.

Arguments

None.

Returns

A Boolean value, `true` if “Hide All Visual Aids” is set, `false` otherwise.

Enabler

None.

dom.getPreventLayerOverlaps()

Availability

Dreamweaver 3.0

Description

Gets the current state of the Prevent Layer Overlaps option.

Arguments

None.

Returns

A Boolean value indicating whether the option is on (`true`) or off (`false`).

Enabler

None.

dom.getShowAutoIndent()

Description

Determines whether or not auto indenting is on in the Code view of the Document window.

Arguments

None.

Returns

Returns `true` if auto indenting is on.

dom.getShowFrameBorders()

Availability

Dreamweaver 3.0

Description

Gets the current state of the View > Frame Borders option.

Arguments

None.

Returns

A Boolean value indicating whether frame borders are visible (`true`) or not (`false`).

Enabler

None.

dom.getShowGrid()

Availability

Dreamweaver 3.0

Description

Gets the current state of the View > Grid > Show option.

Arguments

None.

Returns

A Boolean value indicating whether the grid is visible (`true`) or not (`false`).

Enabler

None.

dom.getShowHeaderView()

Availability

Dreamweaver 3.0

Description

Gets the current state of the View > Head Content option.

Arguments

None.

Returns

A Boolean value indicating whether head content is visible (`true`) or not (`false`).

Enabler

None.

dom.getShowHighlightInvalidHTML()

Availability

Dreamweaver 4.0

Description

Determines whether or not invalid HTML is currently highlighted in the Code view of the Document window.

Arguments

None.

Returns

Returns `true` if invalid HTML is being highlighted.

dom.getShowImageMaps()

Availability

Dreamweaver 3.0

Description

Gets the current state of the View > Image Maps option.

Arguments

None.

Returns

A Boolean value indicating whether image maps are visible (`true`) or not (`false`).

Enabler

None.

dom.getShowInvisibleElements()

Availability

Dreamweaver 3.0

Description

Gets the current state of the View > Invisible Elements option.

Arguments

None.

Returns

A Boolean value indicating whether invisible element markers are visible (`true`) or not (`false`).

Enabler

None.

dom.getShowLayerBorders()

Availability

Dreamweaver 3.0

Description

Gets the current state of the View > Layer Borders option.

Arguments

None.

Returns

A Boolean value indicating whether layer borders are visible (`true`) or not (`false`).

Enabler

None.

dom.getShowLineNumbers()

Availability

Dreamweaver 4.0

Description

Determines whether line numbers are displayed in the Code view.

Arguments

None.

Returns

Returns a Boolean value indicating whether line numbers are shown (`true`) or not (`false`).

dom.getShowRulers()

Availability

Dreamweaver 3.0

Description

Gets the current state of the View > Rulers > Show option.

Arguments

None.

Returns

A Boolean value indicating whether the rulers are visible (`true`) or not (`false`).

Enabler

None.

dom.getShowSyntaxColoring()

Availability

Dreamweaver 4.0

Description

Determines whether or not syntax coloring is on in the Code view of the Document window.

Arguments

None.

Returns

Returns `true` if syntax coloring is on.

dom.getShowTableBorders()

Availability

Dreamweaver 3.0

Description

Gets the current state of the View > Table Borders option.

Arguments

None.

Returns

A Boolean value indicating whether table borders are visible (`true`) or not (`false`).

Enabler

None.

dom.getShowToolbar()

Availability

Dreamweaver 4.0

Description

Determines whether or not the toolbar is being shown.

Arguments

None.

Returns

Returns `true` if the toolbar is displayed; `false` if the toolbar is not displayed.

dom.getShowTracingImage()

Availability

Dreamweaver 3.0

Description

Gets the current state of the View > Tracing Image > Show option.

Arguments

None.

Returns

A Boolean value indicating whether the option is on (`true`) or off (`false`).

Enabler

None.

dom.getShowWordWrap()

Availability

Dreamweaver 4.0

Description

Determines whether or not word wrap is on in the Code view of the Document window.

Arguments

None.

Returns

Returns `true` if word wrap is on.

dom.getSnapToGrid()

Availability

Dreamweaver 3.0

Description

Gets the current state of the View > Grid > Snap To option.

Arguments

None.

Returns

A Boolean value indicating whether grid snapping is on (*true*) or off (*false*).

Enabler

None.

dom.setEditNoFramesContent()

Availability

Dreamweaver 3.0

Description

Turns the Modify > Frameset > Edit NoFrames Content option on (*true*) or off (*false*).

Arguments

bEditNoFrames

Returns

Nothing.

Enabler

`dom.canEditNoFramesContent()`

dom.setHideAllVisualAids()

Availability

Dreamweaver 4.0

Description

Turns off the display of all borders, image maps, and invisible elements, regardless of their individual settings in the View menu.

Arguments

bSet

bSet is a Boolean value; when set to `false`, the previous settings are restored.

Returns

Nothing.

Enabler

None.

dom.setPreventLayerOverlaps()

Availability

Dreamweaver 3.0

Description

Turns the Prevent Layer Overlaps option on (`true`) or off (`false`).

Arguments

bPreventLayerOverlaps

Returns

Nothing.

Enabler

None.

dom.setShowFrameBorders()

Availability

Dreamweaver 3.0

Description

Turns the View > Frame Borders option on (true) or off (false).

Arguments

bShowFrameBorders

Returns

Nothing.

Enabler

None.

dom.setShowGrid()

Availability

Dreamweaver 3.0

Description

Turns the View > Grid > Show option on (true) or off (false).

Arguments

bShowGrid

Returns

Nothing.

Enabler

None.

dom.setShowHeaderView()

Availability

Dreamweaver 3.0

Description

Turns the View > Head Content option on (true) or off (false).

Arguments

bShowHead

Returns

Nothing.

Enabler

None.

dom.setShowHighlightInvalidHTML()

Availability

Dreamweaver 4.0

Description

Turns highlighting of invalid HTML on or off in the Code view of the Document window.

Arguments

bShow

bShow is a Boolean value indicating whether the highlighting of invalid HTML should be visible (true) or not (false).

Returns

Nothing.

dom.setShowImageMaps()

Availability

Dreamweaver 3.0

Description

Turns the View > Image Maps option on (true) or off (false).

Arguments

bShowImageMaps

Returns

Nothing.

Enabler

None.

dom.setShowInvisibleElements()

Availability

Dreamweaver 3.0

Description

Turns the View > Invisible Elements option on (true) or off (false).

Arguments

bViewInvisibleElements

Returns

Nothing.

Enabler

None.

dom.setShowLayerBorders()

Availability

Dreamweaver 3.0

Description

Turns the View > Layer Borders option on (true) or off (false).

Arguments

bShowLayerBorders

Returns

Nothing.

Enabler

None.

dom.setShowLineNumbers()

Availability

Dreamweaver 4.0

Description

Shows or hides the line numbers in the Code view of the Document window.

Arguments

bShow

bShow is a Boolean value indicating whether the line numbers should be visible (true) or not (false).

Returns

Nothing.

dom.setShowRulers()

Availability

Dreamweaver 3.0

Description

Turns the View > Rulers > Show option on (true) or off (false).

Arguments

bShowRulers

Returns

Nothing.

Enabler

None.

`dom.setShowSyntaxColoring()`

Availability

Dreamweaver 4.0

Description

Turns syntax coloring on or off in the Code view of the Document window.

Arguments

bShow

bShow is a Boolean value indicating whether the syntax coloring should be visible (true) or not (false).

Returns

Nothing.

`dom.setShowTableBorders()`

Availability

Dreamweaver 3.0

Description

Turns the View > Table Borders option on (true) or off (false).

Arguments

bShowTableBorders

Returns

Nothing.

Enabler

None.

`dom.setShowToolbar()`

Availability

Dreamweaver 4.0

Description

Shows or hides the Toolbar.

Arguments

bShow

bShow is a Boolean value indicating whether the toolbar should be visible (true) or not (false).

Returns

Nothing.

dom.setShowTracingImage()

Availability

Dreamweaver 3.0

Description

Turns the View > Tracing Image > Show option on (true) or off (false).

Arguments

bShowTracingImage

Returns

Nothing.

Enabler

None.

dom.setShowWordWrap()

Availability

Dreamweaver 4.0

Description

Turns word wrap off or on in the Code view of the Document window.

Arguments

bShow

bShow is a Boolean value indicating whether the line numbers should be visible (true) or not (false).

Returns

Nothing.

dom.setSnapToGrid()

Availability

Dreamweaver 3.0

Description

Turns the View > Grid > Snap To option on (true) or off (false).

Arguments

bSnapToGrid

Returns

Nothing.

Enabler

None.

`dreamweaver.getHideAllFloaters()`

Availability

Dreamweaver 3.0

Description

Gets the current state of the Hide Floating Panels option.

Arguments

None.

Returns

A Boolean value indicating whether the Hide Floating Panels option (`true`) or the Show Floating Panels option (`false`) is available.

Enabler

None.

`dreamweaver.getShowStatusBar()`

Availability

Dreamweaver 3.0

Description

Gets the current state of the View > Status Bar option.

Arguments

None.

Returns

A Boolean value indicating whether the status bar is visible (`true`) or not (`false`).

Enabler

None.

`dreamweaver.htmlInspector.getShowAutoIndent()`

Availability

Dreamweaver 4.0

Description

Determines whether or not auto indenting is on in the Code view of the Code inspector.

Arguments

None.

Returns

Returns `true` if auto indenting is on.

`dreamweaver.htmlInspector.getShowHighlightInvalidHTML()`

Description

Determines whether or not invalid HTML is currently highlighted in the Code view of the Code inspector.

Arguments

None.

Returns

Returns `true` if invalid HTML is currently highlighted.

`dreamweaver.htmlInspector.getShowLineNumbers()`

Availability

Dreamweaver 4.0

Description

Determines whether or not line numbers are being shown in the Code view of the Code inspector.

Arguments

None.

Returns

Returns `true` if line numbers are shown.

`dreamweaver.htmlInspector.getShowSyntaxColoring()`

Availability

Dreamweaver 4.0

Description

Determines whether or not syntax coloring is on in the Code view of the Code inspector.

Arguments

None.

Returns

Returns `true` if syntax coloring is on.

`dreamweaver.htmlInspector.getShowWordWrap()`

Availability

Dreamweaver 4.0

Description

Determines whether or not word wrap is on in the Code view of the Code inspector.

Arguments

None.

Returns

Returns `true` if word wrap is on.

`dreamweaver.htmlInspector.setShowAutoIndent()`

Availability

Dreamweaver 4.0

Description

Turns auto indenting on or off in the Code view of the Code inspector.

Arguments

bShow

bShow is a Boolean value indicating whether the auto indenting should be on (`true`) or off (`false`).

Returns

Nothing.

`dreamweaver.htmlInspector.setShowHighlightInvalidHTML()`

Availability

Dreamweaver 4.0

Description

Turns highlighting of invalid HTML on or off in the Code view of the Code inspector.

Arguments

bShow

bShow is a Boolean value indicating whether the highlighting of invalid HTML should be visible (`true`) or not (`false`).

Returns

Nothing.

`dreamweaver.htmlInspector.setShowLineNumbers()`

Availability

Dreamweaver 4.0

Description

Shows or hides the line numbers in the Code view of the Code inspector.

Arguments

bShow

bShow is a Boolean value indicating whether the line numbers should be visible (true) or not (false).

Returns

Nothing.

`dreamweaver.htmlInspector.setShowSyntaxColoring()`

Availability

Dreamweaver 4.0

Description

Turns syntax coloring on or off in the Code view of the Code inspector.

Arguments

bShow

bShow is a Boolean value indicating whether the syntax coloring should be visible (true) or not (false).

Returns

Nothing.

`dreamweaver.htmlInspector.setShowWordWrap()`

Availability

Dreamweaver 4.0

Description

Turns word wrap off or on in the Code view of the Code inspector.

Arguments

bShow

bShow is a Boolean value indicating whether the word wrapping should be on (true) or off (false).

Returns

Nothing.

`dreamweaver.setHideAllFloaters()`

Availability

Dreamweaver 3.0

Description

Turns on either the Hide Floating Panels option (`true`) or the Show Floating Panels option (`false`).

Arguments

bShowFloatingPalettes

Returns

Nothing.

Enabler

None.

`dreamweaver.setShowStatusBar()`

Availability

Dreamweaver 3.0

Description

Turns the View > Status Bar option on (`true`) or off (`false`).

Arguments

bShowStatusBar

Returns

Nothing.

Enabler

None.

site.getShowDependents()

Availability

Dreamweaver 3.0

Description

Gets the current state of the Show Dependent Files option.

Arguments

None.

Returns

A Boolean value indicating whether dependent files are visible in the site map (`true`) or not (`false`).

Enabler

None.

site.getShowHiddenFiles()

Availability

Dreamweaver 3.0

Description

Gets the current state of the Show Files Marked as Hidden option.

Arguments

None.

Returns

A Boolean value indicating whether hidden files are visible in the site map (`true`) or not (`false`).

Enabler

None.

site.getShowPageTitles()

Availability

Dreamweaver 3.0

Description

Gets the current state of the Show Page Titles option.

Arguments

None.

Returns

A Boolean value indicating whether page titles are visible in the site map (`true`) or not (`false`).

Enabler

None.

site.getShowToolTips()

Availability

Dreamweaver 3.0

Description

Gets the current state of the Tool Tips option.

Arguments

None.

Returns

A Boolean value indicating whether tool tips are visible in the Site window (`true`) or not (`false`).

Enabler

None.

site.setShowDependents()

Availability

Dreamweaver 3.0

Description

Turns the Show Dependent Files option in the site map on (true) or off (false).

Arguments

bShowDependentFiles

Returns

Nothing.

Enabler

None.

site.setShowHiddenFiles()

Availability

Dreamweaver 3.0

Description

Turns the Show Files Marked as Hidden option in the site map on (true) or off (false).

Arguments

bShowHiddenFiles

Returns

Nothing.

Enabler

None.

site.setShowPageTitles()

Availability

Dreamweaver 3.0

Description

Turns the Show Page Titles option in the site map on (true) or off (false).

Arguments

bShowPageTitles

Returns

Nothing.

Enabler

site.canShowPageTitles()

`site.setShowToolTips()`

Availability

Dreamweaver 3.0

Description

Turns the Tool Tips option on (true) or off (false).

Arguments

bShowToolTips

Returns

Nothing.

Enabler

None.

Translation functions

Translation functions deal either directly with translators or with the results of translation. These functions get information about or run a translator, edit content in a locked region, and specify that the translated source should be used when getting and setting selection offsets.

`dom.runTranslator()`

Availability

Dreamweaver 3.0

Description

Runs the specified translator on the document. This function is valid only for the active document.

Arguments

translatorName

translatorName is the name of a translator as it appears in the Translation preferences.

Returns

Nothing.

Enabler

None.

`dreamweaver.editLockedRegions()`

Availability

Dreamweaver 2.0

Description

Depending on the value of the argument, makes locked regions editable or noneditable. By default, locked regions are noneditable; if you try to edit a locked region before specifically making it editable with this function, Dreamweaver beeps and disallows the change.

Note: Editing locked regions can have unintended consequences for library items and templates. It is not recommend that you use this function outside the context of data translators.

Arguments

bAllowEdits

bAllowEdits is a Boolean value indicating that edits are allowed (`true`) or not allowed (`false`). Dreamweaver automatically restores locked regions to their default (noneditable) state when the script that calls this function finishes executing.

Returns

Nothing.

Enabler

None.

`dreamweaver.getTranslatorList()`

Availability

Dreamweaver 3.0

Description

Gets a list of the installed translators.

Arguments

None.

Returns

An array of strings, each representing the name of a translator as it appears in the Translation preferences.

Enabler

None.

`dreamweaver.useTranslatedSource()`

Availability

Dreamweaver 2.0

Description

Specifies that the values returned by `dom.nodeToOffsets()` and `dom.getSelection()` and used by `dom.offsetsToNode()` and `dom.setSelection()` should be offsets into the translated source (the HTML contained in the DOM after a translator is run), not the untranslated source.

Note: This function is relevant only in Property Inspector files.

Arguments

bUseTranslatedSource

The default value of the argument is `false`. Dreamweaver automatically uses the untranslated source for subsequent calls to `dw.getSelection()`, `dw.setSelection()`, `dw.nodeToOffsets()`, and `dw.offsetsToNode()` when the script that calls `dw.useTranslatedSource()` finishes executing, if `dw.useTranslatedSource()` is not explicitly called with an argument of `false` before then.

Returns

Nothing.

Enabler

None.

Layout environment functions

Layout environment functions handle operations related to the settings for working on a document. They affect the source, position, and opacity of the tracing image; get and set the ruler origin and units; turn the grid on and off and change its settings; and play or stop playing plugins.

`dom.getRulerOrigin()`

Availability

Dreamweaver 3.0

Description

Gets the origin of the ruler.

Arguments

None.

Returns

An array of two integers. The first array item is the *x* coordinate of the origin, and the second array item is the *y* coordinate of the origin. Both values are in pixels.

Enabler

None.

`dom.getRulerUnits()`

Availability

Dreamweaver 3.0

Description

Gets the current ruler units.

Arguments

None.

Returns

A string containing one of the following values:

- "in"
- "cm"
- "px"

Enabler

None.

dom.getTracingImageOpacity()

Availability

Dreamweaver 3.0

Description

Gets the opacity setting for the document's tracing image.

Arguments

None.

Returns

A value between 0 and 100, or nothing if no opacity is set.

Enabler

`dom.hasTracingImage()`

dom.loadTracingImage()

Availability

Dreamweaver 3.0

Description

Opens the Select Image Source dialog box. If the user selects an image and clicks OK, the Page Properties dialog box opens with the Tracing Image field filled in.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.playAllPlugins()

Availability

Dreamweaver 3.0

Description

Plays all plugin content in the document.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.playPlugin()

Availability

Dreamweaver 3.0

Description

Plays the selected plugin item.

Arguments

None.

Returns

Nothing.

Enabler

dom.canPlayPlugin()

dom.setRulerOrigin()

Availability

Dreamweaver 3.0

Description

Sets the origin of the ruler.

Arguments

xCoordinate, *yCoordinate*

- *xCoordinate* is a value, expressed in pixels, on the horizontal axis.
- *yCoordinate* is a value, expressed in pixels, on the vertical axis.

Returns

Nothing.

Enabler

None.

dom.setRulerUnits()

Availability

Dreamweaver 3.0

Description

Sets the current ruler units.

Arguments

units

units must be "px", "in", or "cm".

Returns

Nothing.

Enabler

None.

dom.setTracingImagePosition()

Availability

Dreamweaver 3.0

Description

Moves the top left corner of the tracing image to the specified coordinates. If the arguments are omitted, the Adjust Tracing Image Position dialog box appears.

Arguments

x, y

Returns

Nothing.

Enabler

dom.hasTracingImage()

dom.setTracingImageOpacity()

Availability

Dreamweaver 3.0

Description

Sets the opacity of the tracing image.

Arguments

opacityPercentage

opacityPercentage must be a number between 0 and 100.

Returns

Nothing.

Enabler

`dom.hasTracingImage()`

Example

The following code sets the opacity of the tracing image to 30%:

```
dw.getDocumentDOM().setTracingOpacity('30');
```

dom.snapTracingImageToSelection()

Availability

Dreamweaver 3.0

Description

Aligns the top left corner of the tracing image with the top left corner of the current selection.

Arguments

None.

Returns

Nothing.

Enabler

`dom.hasTracingImage()`

dom.stopAllPlugins()

Availability

Dreamweaver 3.0

Description

Stops all plugin content that is currently playing in the document.

Arguments

None.

Returns

Nothing.

Enabler

None.

dom.stopPlugin()

Availability

Dreamweaver 3.0

Description

Stops the selected plugin item.

Arguments

None.

Returns

A Boolean value indicating whether the selection is currently being played with a plugin.

Enabler

dom.canStopPlugin()

dreamweaver.arrangeFloatingPalettes()

Availability

Dreamweaver 3.0

Description

Moves the visible floating panels to their default positions.

Arguments

None.

Returns

Nothing.

Enabler

None.

dreamweaver.showGridSettingsDialog()

Availability

Dreamweaver 3.0

Description

Opens the Grid Settings dialog box.

Arguments

None.

Returns

Nothing.

Enabler

None.

Layout view functions

Layout view functions handle operations that change the layout elements within a document. They affect table, column, and cell settings, including position, properties, and appearance.

`dom.addSpacerToColumn()`

Availability

Dreamweaver 4.0

Description

Creates a one-pixel-high transparent spacer image at the bottom of the given column in the currently selected table. This function fails if the current selection is not a table or if the operation wasn't successful.

Arguments

colNum

colNum is the column at the bottom of which the spacer image is created.

Returns

Nothing.

`dom.createLayoutCell()`

Availability

Dreamweaver 4.0

Description

Creates a layout cell in the current document at the specified position and dimensions, either within an existing layout table or in the area below the existing content on the page. If the cell is being created in an existing layout table, it must not overlap or contain any other layout cells or nested layout tables. If the rectangle is not inside an existing layout table, Dreamweaver tries to create a layout table to house the new cell. This function does not force the document into Layout view. This function fails if the cell can't be created.

Arguments

left, top, width, height

- *left* is the *x* position of the left border of the cell.
- *top* is the *y* position of the top border of the cell.
- *width* is the width of the cell in pixels.
- *height* is the height of the cell in pixels.

Returns

Nothing.

dom.createLayoutTable()

Availability

Dreamweaver 4.0

Description

Creates a layout table in the current document at the specified position and dimensions, either within an existing table or in the area below the existing content on the page. If the table is being created in an existing layout table, it cannot overlap other layout cells or nested layout tables, but can contain other layout cells or nested layout tables. This function does not force the document into Layout view. This function fails if the table can't be created.

Arguments

left, top, width, height

- *left* is the *x* position of the left border of the table.
- *top* is the *y* position of the top border of the table.
- *width* is the width of the table in pixels.
- *height* is the height of the table in pixels.

Returns

Nothing.

dom.doesColumnHaveSpacer()

Availability

Dreamweaver 4.0

Description

Determines whether or not a column contains a spacer image that Dreamweaver generated. This function fails if the current selection is not a table.

Arguments

colNum

colNum is the column to check for a spacer image.

Returns

Returns `true` if the specified column in the currently selected table contains a spacer image that Dreamweaver generated; returns `false` otherwise.

dom.doesGroupHaveSpacers()

Availability

Dreamweaver 4.0

Description

Determines whether or not the currently selected table contains a row of spacer images that Dreamweaver generated. This function fails if the current selection is not a table.

Arguments

None.

Returns

Returns `true` if the table contains a row of spacer images; `false` otherwise.

dom.getClickedHeaderColumn()

Availability

Dreamweaver 4.0

Description

If the user has just clicked a menu button in the header of a table in Layout view, causing the table header menu to appear, this function returns the index of the column the user clicked on. The result is undefined if the table header menu isn't visible.

Arguments

None.

Returns

An integer representing the index of the column.

dom.getShowLayoutTableTabs()

Description

Determines whether or not the current document shows tabs for layout tables while in Layout view.

Arguments

None.

Returns

Returns `true` if the current document displays tabs for layout tables while in Layout view; `false` otherwise.

dom.getShowLayoutView()

Description

Determines the view for the current document, either Layout view or Standard view.

Arguments

None.

Returns

Returns `true` if the current document is in Layout view, `false` if the document is in Standard view.

dom.isColumnAutostretch()

Availability

Dreamweaver 4.0

Description

Determines whether or not a column is set to expand or contract automatically depending on the document size. This function fails if the current selection is not a table.

Arguments

colNum

colNum is the column to be sized automatically or fixed width.

Returns

Returns `true` if the column at the given index in the currently selected table is set to “autostretch”, `false` otherwise.

dom.makeCellWidthsConsistent()

Availability

Dreamweaver 4.0

Description

In the currently selected table, sets the width of each column in the HTML to match the currently rendered width of the column. This function fails if the current selection is not a table or if the operation was not successful.

Arguments

None.

Returns

Nothing.

dom.removeAllSpacers()

Availability

Dreamweaver 4.0

Description

Removes all spacer images generated by Dreamweaver from the currently selected table. This function fails if the current selection is not a table or if the operation was not successful.

Arguments

None.

Returns

Nothing.

dom.removeSpacerFromColumn()

Availability

Dreamweaver 4.0

Description

Removes the spacer image from a specified column and deletes the spacer row if there are no more spacer images that Dreamweaver generated. This function fails if the current selection is not a table or if the operation was not successful.

Arguments

colNum

colNum is the column from which to remove the spacer image.

Returns

Nothing.

dom.setColumnAutostretch()

Availability

Dreamweaver 4.0

Description

Switches a column between automatically sized or fixed width. If *bAutostretch* is true, the column at the given index in the currently selected table is set to “autostretch”; otherwise it’s set to a fixed width at its current rendered width. This function fails if the current selection isn’t a table or if the operation wasn’t successful.

Arguments

colNum, *bAutostretch*

- *colNum* is the column to be sized automatically or a fixed width.
- *bAutostretch* indicates if the column is set to “autostretch” (true) or a fixed width (false).

Returns

Nothing.

dom.setShowLayoutTableTabs()

Availability

Dreamweaver 4.0

Description

Sets the current document to display tabs for layout tables whenever it’s in Layout view. This function does not force the document into Layout view.

Arguments

bShow

bShow indicates whether or not to display tabs for layout tables when the current document is in Layout view. If *bShow* is true, shows tabs; false otherwise.

Returns

Nothing.

`dom.setShowLayoutView()`

Availability

Dreamweaver 4.0

Description

Places the current document in Layout view if *bShow* is true.

Arguments

bShow

bShow is a Boolean that toggles the current document between Layout view and Standard view. If *bShow* is true, the current document switches to Layout view.

Returns

Nothing.

Window functions

Window functions deal with operations related to the Document window and the floating panels. These functions show and hide floating panels, determine which part of the Document window has focus, and set the active document. For operations related specifically to the Site window, see “Site functions” on page 348.

`dom.getFocus()`

Availability

Dreamweaver 3.0

Description

Determines the part of the document that is currently in focus.

Arguments

None.

Returns

One of the following strings:

- "head" if the HEAD area is active
- "body" if the BODY or NOFRAMES area is active
- "frameset" if a frameset or any of its frames is selected
- "none" if the focus is not in the document (for example, if it's in the Property inspector or another floating panel)

Enabler

None.

dom.getView()

Availability

Dreamweaver 4.0

Description

Determines which view is visible.

Arguments

None.

Returns

"design", "code", or "split", depending on the visible view.

dom.getWindowTitle()

Availability

Dreamweaver 3.0

Description

Gets the title of the window that contains the document.

Arguments

None.

Returns

A string containing the text that appears between the TITLE tags in the document, or nothing if the document is not in an open window.

Enabler

None.

dom.setView()

Availability

Dreamweaver 4.0

Description

Shows or hides the Design or Code view to produce a design-only, code-only, or split view.

Arguments

viewString

viewString is the view to produce; it must be one of the following values: design, "code", or "split".

Returns

Nothing.

`dreamweaver.getActiveWindow()`

Availability

Dreamweaver 3.0

Description

Gets the document in the active window.

Arguments

None.

Returns

The document object corresponding to the document in the active window; or, if the document is in a frame, the document object corresponding to the frameset.

Enabler

None.

`dreamweaver.getDocumentList()`

Availability

Dreamweaver 3.0

Description

Gets a list of all the open documents.

Arguments

None.

Returns

An array of document objects, each corresponding to an open Document window. If a Document window contains a frameset, the document object refers to the frameset, not the contents of the frames.

Enabler

None.

`dreamweaver.getFloaterVisibility()`

Availability

Dreamweaver 3.0

Description

Checks whether the specified panel or inspector is visible.

Arguments

floaterName

floaterName is the name of a floating panel. The built-in panels must be referenced using one of the following strings: "objects", "properties", "launcher", "site files", "site map", "library", "css styles", "html styles", "behaviors", "timelines", "html", "layers", "frames", "templates", "history" . If *floaterName* does not match one of the built-in panel names, Dreamweaver searches in the Configuration/Floaters folder for a file called "floaterName.htm".

Returns

true if the floating panel is visible and frontmost, false otherwise or if Dreamweaver cannot find a floating panel with the name *floaterName*.

Enabler

None.

`dreamweaver.getFocus()`

Availability

Dreamweaver 4.0

Description

Determines what part of the application is currently in focus.

Arguments

bAllowFloaters

Returns

One of the following strings:

- "document" if the Document window is in focus.
- "site" if the Site window is in focus.
- "textView" if the Text view is in focus.
- "html" if the Code inspector is in focus.
- floaterName, if bAllowFloaters is true and a floating panel has focus, where floaterName is "objects", "properties", "launcher", "library", "css styles", "html styles", "behaviors", "timelines", "layers", "frames", "templates", or "history".
- (Macintosh) "none" if neither the Site window nor any Document windows are open.

Enabler

None.

`dreamweaver.getPrimaryView()`**Availability**

Dreamweaver 4.0

Description

Determines which view is visible as the primary (on top) view.

Arguments

None.

Returns

"design" or "code", depending on which view is visible or on the top in a split view.

`dreamweaver.getSnapDistance()`**Availability**

Dreamweaver 4.0

Description

Returns the snapping distance in pixels.

Arguments

None.

Returns

An integer representing the snapping distance in pixels. The default is 10 pixels; 0 indicates the Snap feature is off.

dreamweaver.minimizeRestoreAll()

Availability

Dreamweaver 4.0

Description

Minimizes (reduces the window to an icon) or restores all windows in Dreamweaver.

Arguments

bMinimize

bMinimize is a Boolean value. *true* indicates that windows should be minimized; *false* indicates that minimized windows should be restored.

Returns

Nothing.

dreamweaver.setActiveWindow()

Availability

Dreamweaver 3.0

Description

Activates the window containing the specified document.

Arguments

documentObject, {*bActivateFrame*}

- *documentObject* is the object at the root of a document's DOM tree (the value returned by `dreamweaver.getDocumentDOM()`).
- *bActivateFrame*, applicable only if *documentObject* is inside a frameset, is a Boolean value indicating whether to activate the frame that contains the document as well as the window that contains the frameset.

Returns

Nothing.

Enabler

None.

`dreamweaver.setFloaterVisibility()`

Availability

Dreamweaver 3.0

Description

Specifies whether to make a particular floating panel or inspector visible.

Arguments

floaterName, *bIsVisible*

- *floaterName* is the name of a floating panel. The built-in panels must be referenced using one of the following strings: "objects", "properties", "launcher", "site files", "site map", "library", "css styles", "html styles", "behaviors", "timelines", "html", "layers", "frames", "templates", "history". If *floaterName* does not match one of the built-in panel names, Dreamweaver searches in the Configuration/Floaters folder for a file called "floaterName.htm". If Dreamweaver cannot find a floating panel with the name *floaterName*, this function has no effect.
- *bIsVisible* is a Boolean value indicating whether to make the floating panel visible.

Returns

Nothing.

Enabler

None.

`dreamweaver.setPrimaryView()`

Availability

Dreamweaver 4.0

Description

Displays the specified view at the top of the Document window.

Arguments

viewString

viewString is the view to bring to the top of the Document window; it can be one of the following values: "design" or "code".

Returns

Nothing.

`dreamweaver.setSnapDistance()`

Availability

Dreamweaver 4.0

Description

Sets the snapping distance in pixels (0 to turn it off; default is 10 pixels)

Arguments

snapDistance

snapDistance is an integer representing the snapping distance in pixels. The default is 10 pixels. Specify 0 to turn the Snap feature off.

Returns

Nothing.

`dreamweaver.showProperties()`

Availability

Dreamweaver 3.0

Description

Makes the Property inspector visible and gives it focus.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.toggleFloater()`

Availability

Dreamweaver 3.0

Description

Shows, hides, or brings to the front the specified panel or inspector.

Note: This function is meaningful only in the `menus.xml` file. To show, bring forward, or hide a floating panel, use `dw.setFloaterVisibility()`.

Arguments

floaterName

floaterName is the name of the window. If the floating panel name is `reference`, the visible/invisible state of the Reference panel can be updated by the user's selection in the Code view. All other panels track the selection all the time, but the Reference panel tracks the selection in the Code View only when the user invokes tracking.

Returns

Nothing.

Enabler

None.

`dreamweaver.updateReference()`

Availability

Dreamweaver 4.0

Description

Updates the Reference floating panel. If the Reference floating panel is not visible, makes it visible and then updates it.

Arguments

None.

Returns

Nothing.

Deprecated functions

Deprecated functions work, but have been superseded by newer features in Dreamweaver. You should use the newer alternatives, because support for the deprecated functions may be withdrawn in future Dreamweaver versions.

`dreamweaver.getBehaviorEvent()`

Availability

Dreamweaver 1.2, deprecated in 2.0 because actions are now chosen before events.

Description

In a Behavior action file, gets the event that triggers this action.

Arguments

None.

Returns

A string representing the event. This is the same string that is passed as an argument (*event*) to the `canAcceptBehavior()` function.

Example

The following instance of `getBehaviorEvent()` is from the `initializeUI()` function in the Drag Layer action file. The role of the following snippet is similar to that of `canAcceptBehavior()`—that is, it checks whether the selected event is appropriate for the selected action. Such a construct is more useful than `canAcceptBehavior()`, because it lets you offer the user some guidance about which events are suitable for calling the selected action. `canAcceptBehavior()` can make the action unavailable in the Actions pop-up menu only if the user chooses an inappropriate event.

```
theEvent = dreamweaver.getBehaviorEvent().toLowerCase();
CANBEAPPLIED = (theEvent != "onmousedown" && theEvent != "onmousemove");
if (CANBEAPPLIED) {
    [display the Drag Layer UI]
} else{
    [display a helpful message that tells the user which events are appropriate for the Drag Layer action.]
}
```

`dreamweaver.getObjectRefs()`

Availability

Dreamweaver 1.0

Description

Scans the specified documents for instances of the specified tags (or, if no tags are specified, for all tags in the document) and formulates browser-specific references to them. This function is equivalent to calling `getElementsByTagName()` and then calling `dreamweaver.getElementRef()` for each tag in the `odelist`.

Arguments

NSorIE, *sourceDoc*, {*tag1*}, {*tag2*},...{*tagN*}

- *NSorIE* must be either "NS 4.0" or "IE 4.0". The DOM and rules for nested references differ in Navigator 4.0 and Internet Explorer 4.0. This argument specifies for which browser to return a valid reference.
- *sourceDoc* must be "document", "parent", "parent.frames[*number*]", "parent.frames['*frameName*']", or a URL. `document` specifies the document that has the focus and contains the current selection. `parent` specifies the parent frameset (if the currently selected document is in a frame), and `parent.frames[number]` and `parent.frames['frameName']` specify a document that is in a particular frame within the frameset containing the current document. If the argument is a relative URL, it is relative to the extension file.
- The third and subsequent arguments, if supplied, are the names of tags (for example, "IMG", "FORM", "HR").

Returns

An array of strings, each a valid JavaScript reference to a named instance of the requested tag type in the specified document (for example, "`document.myLayer.document.myImage`") for the specified browser.

- Dreamweaver returns correct references for Internet Explorer for A, AREA, APPLET, EMBED, DIV, SPAN, INPUT, SELECT, OPTION, TEXTAREA, OBJECT, and IMG tags.
- Dreamweaver returns correct references for Navigator for A, AREA, APPLET, EMBED, LAYER, ILAYER, SELECT, OPTION, TEXTAREA, OBJECT, and IMG tags, and for absolutely positioned DIV and SPAN tags. For DIV and SPAN tags that are not absolutely positioned, Dreamweaver returns "cannot reference <*tag*>".
- Dreamweaver does not return references for unnamed objects. If an object does not contain either a NAME or an ID attribute, then Dreamweaver returns "unnamed <*tag*>". If the browser does not support a reference by name, Dreamweaver references the object by index (for example, `document.myform.applets[3]`).
- Dreamweaver does return references for named objects contained in unnamed forms and layers (for example, `document.forms[2].myCheckbox`).

When the same list of arguments is passed to `getObjectTags()`, the two functions return arrays of the same length and with parallel content.

Example

`dreamweaver.getObjectRefs("NS 4.0", "document", "IMG")`, depending on the contents of the active document, might return an array with the following items:

- `"document.bullet"`
- `"document.layers['headerLayer'].document.header"`
- `"document.photoLayer.document.headshot"`

`dreamweaver.getObjectTags()`

Availability

Dreamweaver1.0

Description

Scans the specified document for instances of the specified tags or, if no tags are specified, for all tags in the document. This function is equivalent to calling `getElementsByTagName()` and then getting `outerHTML` for each element in the `odelist`.

Arguments

sourceDoc, {*tag1*}, {*tag2*},...{*tagN*}

- *sourceDoc* must be `"document"`, `"parent"`, `"parent.frames[number]"`, `"parent.frames['frameName']"`, or a URL. *document* specifies the document that has the focus and contains the current selection. *parent* specifies the parent frameset (if the currently selected document is in a frame), and *parent.frames[number]* and *parent.frames['frameName']* specify a document that is in a particular frame within the frameset containing the current document. If the argument is a relative URL, it is relative to the extension file.
- The second and subsequent arguments, if supplied, are the names of tags (for example, `"IMG"`, `"FORM"`, `"HR"`).

Returns

An array of strings, each the HTML source code for an instance of the requested tag type in the specified document.

- If one of the *tag* arguments is `LAYER`, the function returns all `LAYER` and `ILAYER` tags and all absolutely positioned `DIV` and `SPAN` tags.
- If one of the *tag* arguments is `INPUT`, the function returns all form elements. To get a particular type of form element, specify `INPUT/TYPE`, where *TYPE* is `button`, `text`, `radio`, `checkbox`, `password`, `textarea`, `select`, `hidden`, `reset`, or `submit`.

When the same list of arguments is passed to `getObjectRefs()`, the two functions return arrays of the same length.

Example

`dreamweaver.getObjectTags("document", "IMG")`, depending on the contents of the active document, might return an array with the following items:

- `''`
- `''`
- `''`

`dreamweaver.getSelection()`

Availability

Dreamweaver 2.0, deprecated in 3.0 in favor of `dom.getSelection()`.

Description

Gets the selection in the current document, expressed as byte offsets into the document's HTML source code.

Arguments

None.

Returns

An array containing two integers. The first integer is the byte offset of the beginning of the selection; the second integer is the byte offset of the end of the selection. If the two numbers are the same, then the current selection is an insertion point.

`dreamweaver.libraryPalette.deleteSelectedItem()`

Availability

Dreamweaver 3.0, deprecated in Dreamweaver 4.0 in favor of using `dw.assetPalette.setSelectedCategory("library")`, then calling `dw.assetPalette.removeFromFavorites`.

Description

Removes the selected library item from the Library panel and deletes its associated LBI file from the Library folder at the root of the current site. Instances of the deleted item may still exist in pages throughout the site.

Arguments

None.

Returns

Nothing.

Enabler

None.

dreamweaver.libraryPalette.getSelectedItem()

Availability

Dreamweaver 3.0, deprecated in 4.0 in favor of `dw.assetPalette.getSelectedItems()`.

Description

Gets the path of the selected library item.

Arguments

None.

Returns

A string containing the path of the library item, expressed as a file:// URL.

Enabler

None.

dreamweaver.libraryPalette.newFromDocument()

Availability

Dreamweaver 3.0, deprecated in Dreamweaver 4.0 in favor of using `dw.assetPalette.setSelectedCategory("library")`, then calling `dw.assetPalette.newAsset()`.

Description

Creates a new library item based on the selection in the current document.

Arguments

bReplaceCurrent

bReplaceCurrent is a Boolean value indicating whether to replace the selection with an instance of the newly created library item.

Returns

Nothing.

Enabler

None.

`dreamweaver.libraryPalette.recreateFromDocument()`

Availability

Dreamweaver 3.0, deprecated in Dreamweaver 4.0 in favor of `dw.assetPalette.recreateLibraryFromDocument()`.

Description

Creates an LBI file for the selected instance of a library item in the current document. This function is equivalent to clicking Recreate in the Property inspector.

Arguments

None.

Returns

Nothing.

Enabler

None.

`dreamweaver.libraryPalette.renameSelectedItem()`

Availability

Dreamweaver 3.0, deprecated in Dreamweaver 4.0 in favor of using `dw.assetPalette.setSelectedCategory("library")`, then calling `dw.assetPalette.renameNickname()`.

Description

Turns the name of the selected library item into an edit field, allowing the user to rename the selection.

Arguments

None.

Returns

Nothing.

Enabler

None.

dreamweaver.nodeToOffsets()

Availability

Dreamweaver 2.0, deprecated in 3.0 in favor of `dom.nodeToOffsets()`.

Description

Gets the position of a specific node in the DOM tree, expressed as byte offsets into the document's HTML source code.

Arguments

node

node must be a tag, comment, or range of text that is a node in the tree returned by `dreamweaver.getDocumentDOM()`.

Returns

An array containing two integers. The first integer is the byte offset of the beginning of the tag, text, or comment; the second integer is the byte offset of the end of the node.

Example

The following code selects the first image object in the current document:

```
var theDOM = dreamweaver.getDocumentDOM("document");
var theImg = theDOM.images[0];
var offsets = dom.nodeToOffsets(theImg);
dom.setSelection(offsets[0], offsets[1]);
```

dreamweaver.templatePalette.getSelectedTemplate()

Availability

Dreamweaver 3.0, deprecated in 4.0 in favor of `dw.assetPalette.getSelectedItems()`.

Description

Gets the path of the selected template.

Arguments

None.

Returns

A string containing the path of the template, expressed as a file:// URL.

Enabler

None.

`dreamweaver.offsetsToNode()`

Availability

Dreamweaver 2.0, deprecated in 3.0 in favor of `dom.offsetsToNode()`.

Description

Gets the object in the DOM tree that completely contains the range of characters between the specified begin and end points.

Arguments

offsetBegin, offsetEnd

The arguments are the begin and end points, respectively, of a range of characters, expressed as byte offsets into the document's HTML source code.

Returns

The tag, text, or comment object that completely contains the specified range of characters.

Example

The following code displays an alert if the selection is an image:

```
var offsets = dreamweaver.getSelection();
var theSelection = dreamweaver.offsetsToNode(offsets[0], ↵
offsets[1]);
if (theSelection.nodeType == Node.ELEMENT_NODE && ↵
theSelection.tagName == 'IMG'){
    alert('The current selection is an image.');
```

`dreamweaver.popupCommand()`

Availability

Dreamweaver 2.0, deprecated in 3.0 in favor of `dreamweaver.runCommand()`.

Description

Executes the specified command. To the user, the effect is the same as choosing the command from a menu; if a dialog box is associated with the command, it appears. This function provides the ability to call a command from another extension file. It blocks other edits until the user dismisses the dialog box.

Note: This function can be called only within `objectTag()` or in any script in a command or Property Inspector file.

Arguments

commandFile

commandFile is the name of a command file within the Configuration/Commands folder (for example, "Format Table.htm").

Returns

Nothing.

`dreamweaver.setSelection()`

Availability

Dreamweaver 2.0, deprecated in 3.0 in favor of `dom.setSelection()`.

Description

Sets the selection in the current document. This function can move the selection only within the current document; it cannot change the focus to a different document.

Arguments

offsetBegin, offsetEnd

The arguments are the begin and end points, respectively, for the new selection, expressed as byte offsets into the document's HTML source code. If the two numbers are the same, the new selection is an insertion point. If the new selection is not a valid HTML selection, it is expanded to include the characters in the first valid HTML selection. For example, if *offsetBegin* and *offsetEnd* define the range `SRC="myImage.gif"` within ``, the selection expands to include the entire `IMG` tag.

Returns

Nothing.

`dreamweaver.templatePalette.deleteSelectedTemplate()`

Availability

Dreamweaver 3.0, deprecated in Dreamweaver 4.0 in favor of using `dw.assetPalette.setSelectedCategory("templates")`, then calling `dw.assetPalette.removeFromFavorites()`.

Description

Deletes the selected template from the templates folder.

Arguments

None.

Returns

Nothing.

Enabler

None.

dreamweaver.templatePalette.renameSelectedTemplate()

Availability

Dreamweaver 3.0, deprecated in Dreamweaver 4.0 in favor of using `dw.assetPalette.setSelectedCategory("templates")`, then calling `dw.assetPalette.renameNickname()`.

Description

Turns the name of the selected template into an edit field, allowing the user to rename the selection.

Arguments

None.

Returns

Nothing.

Enabler

None.

Enablers

Enabler functions determine whether to enable menu items based on whether Dreamweaver can perform specific operations in the current context. The function specifications describe the general circumstances under which each function returns `true`. However, the descriptions are not intended to be comprehensive and may exclude some cases in which the function would return `false`.

dom.canAlign()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform an Align Left, Align Right, Align Top, or Align Bottom operation.

Arguments

None.

Returns

A Boolean value indicating whether two or more layers or hotspots are selected.

dom.canApplyTemplate()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform an Apply To Page operation. This function is valid only for the active document.

Arguments

None.

Returns

A Boolean value indicating whether the document is not a library item or a template, and that the selection is not within the NOFRAMES tag.

dom.canArrange()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Bring to Front or Move to Back operation.

Arguments

None.

Returns

A Boolean value indicating whether a hotspot is selected.

dom.canClipCopyText()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Copy as Text operation.

Arguments

None.

Returns

A Boolean value indicating whether the selection is a range (that is, not an insertion point).

dom.canClipPaste()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Paste operation.

Arguments

None.

Returns

A Boolean value indicating whether the clipboard contains any content that can be pasted into Dreamweaver.

dom.canClipPasteText()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Paste as Text operation.

Arguments

None.

Returns

A Boolean value indicating whether the clipboard contains any content that can be pasted into Dreamweaver as text.

dom.canConvertLayersToTable()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Convert Layers to Table operation.

Arguments

None.

Returns

A Boolean value indicating whether all content in the BODY of the document is contained within layers.

dom.canConvertTablesToLayers()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Convert Tables to Layers operation.

Arguments

None.

Returns

A Boolean value indicating whether all the content in the BODY of the document is contained within tables, and the document is not based on a template.

dom.canDecreaseColspan()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Decrease Colspan operation.

Arguments

None.

Returns

A Boolean value indicating whether the current cell has a COLSPAN attribute, and whether that attribute's value is greater than or equal to 2.

dom.canDecreaseRowspan()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Decrease Rowspan operation.

Arguments

None.

Returns

A Boolean value indicating whether the current cell has a ROWSPAN attribute, and whether that attribute's value is greater than or equal to 2.

dom.canDeleteTableColumn()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Delete Column operation.

Arguments

None.

Returns

A Boolean value indicating whether the insertion point is inside a cell, or if a cell or column is selected.

dom.canDeleteTableRow()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Delete Row operation.

Arguments

None.

Returns

A Boolean value indicating whether the insertion point is inside a cell, or if a cell or row is selected.

dom.canEditNoFramesContent()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform an Edit No Frames Content operation.

Arguments

None.

Returns

A Boolean value indicating whether the current document is a frameset or within a frameset.

dom.canIncreaseColspan()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform an Increase Colspan operation.

Arguments

None.

Returns

A Boolean value indicating whether there are any cells to the right of the current cell.

dom.canIncreaseRowspan()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform an Increase Rowspan operation.

Arguments

None.

Returns

A Boolean value indicating whether there are any cells below the current cell.

dom.canInsertTableColumns()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform an Insert Column(s) operation.

Arguments

None.

Returns

A Boolean value indicating whether the selection is inside a table. This function returns `false` if the selection is an entire table.

dom.canInsertTableRows()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform an Insert Row(s) operation.

Arguments

None.

Returns

A Boolean value indicating whether the selection is inside a table. This function returns `false` if the selection is an entire table.

dom.canMakeNewEditableRegion()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a New Editable Region operation.

Arguments

None.

Returns

A Boolean value indicating whether the current document is a template (.dwt) file.

dom.canMarkSelectionAsEditable()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Mark Selection as Editable operation.

Arguments

None.

Returns

A Boolean value indicating whether there is a selection, and whether the current document is a template (.dwt) file.

dom.canMergeTableCells()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Merge Cells operation.

Arguments

None.

Returns

A Boolean value indicating whether the selection is a rectangular grouping of table cells.

dom.canPlayPlugin()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Play operation. This function is valid only for the active document.

Arguments

None.

Returns

A Boolean value indicating whether the selection can be played with a plugin.

dom.canRedo()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Redo operation.

Arguments

None.

Returns

A Boolean value indicating whether any steps remain to redo.

dom.canRemoveEditableRegion()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform an Unmark Editable Region operation.

Arguments

None.

Returns

A Boolean value indicating whether the current document is a template.

dom.canSelectTable()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Select Table operation.

Arguments

None.

Returns

A Boolean value indicating whether the insertion point or selection is within a table.

dom.canSetLinkHref()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can change the link around the current selection or create one if necessary.

Arguments

None.

Returns

A Boolean value indicating whether the selection is an image, text, or an insertion point inside a link. A text selection is defined as a selection for which the text Property inspector would appear.

dom.canShowListPropertiesDialog()

Availability

Dreamweaver 3.0

Description

Determines whether Dreamweaver can show the List Properties dialog box.

Arguments

None.

Returns

A Boolean value indicating whether the selection is within an LI tag.

Enabler

None.

dom.canSplitFrame()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Split Frame [Left | Right | Up | Down] operation.

Arguments

None.

Returns

A Boolean value indicating whether the selection is within a frame.

dom.canSplitTableCell()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Split Cell operation.

Arguments

None.

Returns

A Boolean value indicating whether the insertion point is inside a table cell or the selection is a table cell.

dom.canStopPlugin()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Stop operation.

Arguments

None.

Returns

A Boolean value indicating whether the selection is currently being played with a plugin.

dom.canUndo()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform an Undo operation.

Arguments

None.

Returns

A Boolean value indicating whether any steps remain to undo.

dom.hasTracingImage()

Availability

Dreamweaver 3.0

Description

Checks whether the document has a tracing image.

Arguments

None.

Returns

A Boolean value indicating whether the document has a tracing image.

`dreamweaver.assetPalette.canEdit()`

Availability

Dreamweaver 4.0

Description

Enables menu items in the Assets panel for editing.

Arguments

None.

Returns

Returns `true` if the asset can be edited or `false` if not. Will return `false` for colors and URLs in the Site list, and will return `false` for a multiple selection of colors and URLs in the Favorites list.

`dreamweaver.assetPalette.canInsertOrApply()`

Availability

Dreamweaver 4.0

Description

Determines if the selected elements can be inserted or applied. Returns `true` or `false` so the menu items can be enabled or disabled for insertion or application.

Arguments

None.

Returns

Returns `false` if the current page is a template, and the current category is “Templates”. Returns `false` if no document is open. Returns `false` if a library item is selected in the document and the current category is “Library”. Otherwise returns `true`.

`dreamweaver.canClipCopy()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Copy operation.

Arguments

None.

Returns

A Boolean value indicating whether there is anything selected that can be copied to the clipboard.

`dreamweaver.canClipCut()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Cut operation.

Arguments

None.

Returns

A Boolean value indicating whether there is anything selected that can be cut to the clipboard.

`dreamweaver.canClipPaste()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Paste operation.

Arguments

None.

Returns

A Boolean value indicating whether the clipboard contains any content that can be pasted into the current document or the active pane in the Site window; or, on the Macintosh, an edit field in a floating panel or dialog box.

`dreamweaver.canDeleteSelection()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can delete the current selection. Depending on the window that has focus, the deletion may occur in the Document window or the Site window; or, on the Macintosh, in an edit field in a dialog box or floating panel.

Arguments

None.

Returns

A Boolean value indicating whether the selection is a range (that is, not an insertion point).

`dreamweaver.canExportCSS()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform an Export CSS Styles operation.

Arguments

None.

Returns

A Boolean value indicating whether the document contains any class styles defined in the HEAD.

`dreamweaver.canFindNext()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Find Next operation.

Arguments

None.

Returns

A Boolean value indicating whether a search pattern has already been established.

`dreamweaver.canOpenInFrame()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform an Open in Frame operation.

Arguments

None.

Returns

A Boolean value indicating whether the selection or insertion point is within a frame.

`dreamweaver.canPlayRecordedCommand()`

Availability

Dreamweaver 3.0

Description

Determines whether Dreamweaver can perform a Play Recorded Command operation.

Arguments

None.

Returns

A Boolean value indicating whether there is an active document and a previously recorded command that can be played.

`dreamweaver.canRedo()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Redo operation in the current context.

Arguments

None.

Returns

A Boolean value indicating whether any operations can be undone.

`dreamweaver.canRevertDocument()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Revert (to the last-saved version) operation.

Arguments

documentObject

documentObject is the object at the root of a document's DOM tree (the value returned by `dreamweaver.getDocumentDOM()`).

Returns

A Boolean value indicating whether the document is in an unsaved state and a saved version of the document exists on a local drive.

`dreamweaver.canSaveAll()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Save All operation.

Arguments

None.

Returns

A Boolean value indicating whether one or more unsaved documents are open.

`dreamweaver.canSaveDocument()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Save operation on the specified document.

Arguments

documentObject

documentObject is the object at the root of a document's DOM tree (the value returned by `dreamweaver.getDocumentDOM()`).

Returns

A Boolean value indicating whether the document has any unsaved changes.

`dreamweaver.canSaveDocumentAsTemplate()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Save As Template operation on the specified document.

Arguments

documentObject

documentObject is the object at the root of a document's DOM tree (the value returned by `dreamweaver.getDocumentDOM()`).

Returns

A Boolean value indicating whether the document can be saved as a template.

`dreamweaver.canSaveFrameset()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Save Frameset operation on the specified document.

Arguments

documentObject

documentObject is the object at the root of a document's DOM tree (the value returned by `dreamweaver.getDocumentDOM()`).

Returns

A Boolean value indicating whether the document is a frameset with unsaved changes.

`dreamweaver.canSaveFramesetAs()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Save Frameset As operation on the specified document.

Arguments

documentObject

documentObject is the object at the root of a document's DOM tree (the value returned by `dreamweaver.getDocumentDOM()`).

Returns

A Boolean value indicating whether the document is a frameset.

`dreamweaver.canSelectAll()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Select All operation.

Arguments

None.

Returns

A Boolean value indicating whether a Select All operation can be performed.

`dreamweaver.canShowFindDialog()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Find operation.

Arguments

None.

Returns

A Boolean value indicating whether a Site window or a Document window is open. This function returns `false` when the selection is in the HEAD.

`dreamweaver.canUndo()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform an Undo operation in the current context.

Arguments

None.

Returns

A Boolean value indicating whether any operations can be undone.

`dreamweaver.isRecording()`

Availability

Dreamweaver 3.0

Description

Reports whether Dreamweaver is currently recording a command.

Arguments

None.

Returns

A Boolean value indicating whether Dreamweaver is recording a command.

`dreamweaver.htmlStylePalette.canEditSelection()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can edit, delete, or duplicate the selection in the HTML Styles panel.

Arguments

None.

Returns

A Boolean value. This function returns `false` if no style is selected or if one of the “clear” styles is selected.

`dreamweaver.timelineInspector.canAddFrame()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform an Add Frame operation.

Arguments

None.

Returns

A Boolean value indicating whether the Timelines panel has any animation bars or behaviors.

`dreamweaver.timelineInspector.canAddKeyFrame()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform an Add Keyframe operation.

Arguments

None.

Returns

A Boolean value indicating whether the selection in the Timelines panel is part of an animation bar.

`dreamweaver.timelineInspector.canChangeObject()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Change Object operation.

Arguments

None.

Returns

A Boolean value indicating whether the selection in the Timelines panel is part of an animation bar.

`dreamweaver.timelineInspector.canRemoveBehavior()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Remove Behavior operation.

Arguments

None.

Returns

A Boolean value indicating whether the selection in the Timelines panel is a behavior.

`dreamweaver.timelineInspector.canRemoveFrame()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Remove Frame operation.

Arguments

None.

Returns

A Boolean value indicating whether the Timelines panel has any animation bars or behaviors.

`dreamweaver.timelineInspector.canRemoveKeyFrame()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Remove Keyframe operation.

Arguments

None.

Returns

A Boolean value indicating whether the current frame in the Timelines panel is a keyframe.

`dreamweaver.timelineInspector.canRemoveObject()`

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Remove Object operation.

Arguments

None.

Returns

A Boolean value indicating whether the Timelines panel has any animation bars.

`site.browseDocument()`

Availability

Dreamweaver 4.0

Description

Opens all selected documents in a browser window, as in the Preview in Browser command.

Arguments

browserName

browserName is the name of a browser as defined in the Preview in Browser preferences. If omitted, this argument defaults to the user's primary browser.

Returns

Nothing.

site.canAddLink()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform an Add Link to [Existing File | New File] operation.

Arguments

None.

Returns

A Boolean value indicating that the selected document in the site map is an HTML file.

site.canChangeLink()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Change Link operation.

Arguments

None.

Returns

A Boolean value indicating that an HTML or Flash file links to the selected file in the site map.

site.canCheckIn()

Availability

Dreamweaver 3.0

Description

Determines whether Dreamweaver can perform a Check In operation.

Arguments

siteOrURL

siteOrURL must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.

Returns

A Boolean value indicating whether all of the following conditions are true:

- A remote site has been defined.
- If a Document window has focus, the file has been saved in a local site; or, if the Site window has focus, one or more files or folders is selected.
- Check in/check out is turned on.

site.canCheckOut()

Availability

Dreamweaver 3.0

Description

Determines whether Dreamweaver can perform a Check Out operation on the specified file or files.

Arguments

siteOrURL

siteOrURL must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.

Returns

A Boolean value indicating whether all of the following conditions are true:

- A remote site has been defined.
- If a Document window has focus, the file is part of a local site and is not already checked out; or, if the Site window has focus, one or more files or folders is selected and at least one of the selected files is not already checked out.
- Check in/check out is turned on.

site.canConnect()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can connect to the remote site.

Arguments

None.

Returns

A Boolean value indicating whether the current remote site is an FTP site.

site.canFindLinkSource()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Find Link Source operation.

Arguments

None.

Returns

A Boolean value indicating that the selected link in the site map is not the home page.

site.canGet()

Availability

Dreamweaver 3.0

Description

Determines whether Dreamweaver can perform a Get operation.

Arguments

siteOrURL

siteOrURL must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.

Returns

If the argument is "site", Boolean value indicating whether one or more files or folders is selected in the Site window and a remote site has been defined. If the argument is a URL, a Boolean value indicating whether the document belongs to a site for which a remote site has been defined.

site.canLocateInSite()

Availability

Dreamweaver 3.0

Description

Determines whether Dreamweaver can perform a Locate in Local Site or Locate in Remote Site operation (depending on the argument).

Arguments

localOrRemote, *siteOrURL*

- *localOrRemote* must be either "local" or "remote".
- *siteOrURL* must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.

Returns

One of the following values:

- If the first argument is "local" and the second argument is a URL, a Boolean value indicating whether the document belongs to a site.
- If the first argument is "remote" and the second argument is a URL, a Boolean value indicating whether the document belongs to a site for which a remote site has been defined, and, if the server type is Local/Network, whether the drive is mounted.
- If the second argument is "site", a Boolean value indicating whether both panes contain site files (not the site map) and whether the selection is in the opposite pane from the argument.

site.canMakeEditable()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Turn Off Read Only operation.

Arguments

None.

Returns

A Boolean value indicating whether one or more of the selected files is locked.

site.canMakeNewFileOrFolder()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a New File or New Folder operation in the Site window.

Arguments

None.

Returns

A Boolean value indicating whether any files are visible in the selected pane of the Site window.

site.canOpen()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can open the files or folders that are currently selected in the Site window.

Arguments

None.

Returns

A Boolean value indicating whether any files or folders are selected in the Site window.

site.canPut()

Availability

Dreamweaver 3.0

Description

Determines whether Dreamweaver can perform a Put operation.

Arguments

siteOrURL

siteOrURL must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.

Returns

If the argument is "site", a Boolean value indicating whether any files or folders are selected in the Site window and a remote site has been defined. If the argument is a URL, a Boolean value indicating whether the document belongs to a site for which a remote site has been defined.

site.canRecreateCache()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Recreate Site Cache operation.

Arguments

None.

Returns

A Boolean value indicating whether the Use Cache To Speed Link Updates option is enabled for the current site.

site.canRefresh()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Refresh [Local | Remote] operation.

Arguments

localOrRemote

localOrRemote must be either "local" or "remote".

Returns

true if *localOrRemote* is "local"; otherwise, a Boolean value indicating whether a remote site has been defined.

site.canRemoveLink()

Availability

Dreamweaver 3.0

Description

Checks whether Dreamweaver can perform a Remove Link operation.

Arguments

None.

Returns

A Boolean value indicating that an HTML or Flash file links to the selected file in the site map.

site.canSetLayout()

Availability

Dreamweaver 3.0

Description

Determines whether Dreamweaver can perform a Layout operation.

Arguments

None.

Returns

A Boolean value indicating whether the site map is visible.

site.canSelectAllCheckedOutFiles()

Availability

Dreamweaver 4.0

Description

Determines whether the current working site has check in/check out enabled.

Arguments

None.

Returns

A Boolean value; `true` if the site allows check in/check out, otherwise `false`.

site.canSelectNewer()

Availability

Dreamweaver 3.0

Description

Determines whether Dreamweaver can perform a Select Newer [Remote | Local] operation.

Arguments

localOrRemote

localOrRemote must be either "local" or "remote".

Returns

A Boolean value indicating whether the document belongs to a site for which a remote site has been defined.

site.canShowPageTitles()

Availability

Dreamweaver 3.0

Description

Determines whether Dreamweaver can perform a Show Page Titles operation.

Arguments

None.

Returns

A Boolean value indicating whether the site map is visible.

site.canSynchronize()

Availability

Dreamweaver 3.0

Description

Determines whether Dreamweaver can perform a Synchronize operation.

Arguments

None.

Returns

A Boolean value indicating whether a remote site has been defined.

site.canUndoCheckOut()

Availability

Dreamweaver 3.0

Description

Determines whether Dreamweaver can perform an Undo Checkout operation.

Arguments

siteOrURL

siteOrURL must be the keyword "site", indicating that the function should act on the selection in the Site window, or the URL for a single file.

Returns

A Boolean value indicating whether the specified file or at least one of the selected files is checked out (by any user).

site.canViewAsRoot()

Availability

Dreamweaver 3.0

Description

Determines whether Dreamweaver can perform a View as Root operation.

Arguments

None.

Returns

A Boolean value indicating whether the specified file is an HTML or Flash file.

site.SelectAllCheckedOutFiles()

Availability

Dreamweaver 4.0

Description

Selects all files in the local site that have been checked out.

Arguments

None.

Returns

Nothing.

INDEX



- addBehavior()
 - dom.addBehavior() 216
 - dreamweaver.timelineInspector.addBehavior() 402
- addFrame() 403
- addKeyframe() 403
- addLinkToExistingFile() 349
- addLinkToNewFile() 349
- addObject() 403
- addSpacerToColumn() 441
- addTimeline() 404
- alert() 14
- align() 316
- applyBehavior() 90
- applyCharacterMarkup() 268
- applyCSSStyle() 236
- applyFontMarkup() 268
- applyHTMLStyle() 300
- applyTemplate() 320
- appName property 18
- appVersion property 18
- arguments
 - optional 206
 - passed from menuItem 46
 - receiveArguments() 49
- arrange() 317
- arrangeFloatingPalettes() 440
- array object 14
- arrowDown() 308, 373
- arrowLeft() 308, 374
- arrowRight() 309, 374
- arrowUp() 309, 375
- assetPalette.addToFavoritesFromDocument(),
 - dreamweaver.assetPalette.addToFavoritesFromDocument() 208
- assetPalette.addToFavoritesFromSiteAssets(),
 - dreamweaver.assetPalette.addToFavoritesFromSiteAssets() 208
- assetPalette.addToFavoritesFromSiteWindow(),
 - dreamweaver.assetPalette.addToFavoritesFromSiteWindow() 209
- assetPalette.canEdit(),
 - dreamweaver.assetPalette.canEdit() 476
- assetPalette.canInsertOrApply(),
 - dreamweaver.assetPalette.canInsertOrApply() 476
- assetPalette.copyToSite(),
 - dreamweaver.assetPalette.copyToSite() 209
- assetPalette.edit(),
 - dreamweaver.assetPalette.edit() 210
- assetPalette.getSelectedCategory(),
 - dreamweaver.assetPalette.getSelectedCategory() 210
- assetPalette.getSelectedItems(),
 - dreamweaver.assetPalette.getSelectedItems() 211
- assetPalette.getSelectedView(),
 - dreamweaver.assetPalette.getSelectedView() 211
- assetPalette.insertOrApply(),
 - dreamweaver.assetPalette.insertOrApply() 212
- assetPalette.locateInSite(),
 - dreamweaver.assetPalette.locateInSite() 212
- assetPalette.newAsset(),
 - dreamweaver.assetPalette.newAsset() 213
- assetPalette.newFolder(),
 - dreamweaver.assetPalette.newFolder() 213
- assetPalette.recreateLibraryFromDocument(),
 - dreamweaver.assetPalette.recreateLibraryFromDocument() 213
- assetPalette.refreshSiteAssets(),
 - dreamweaver.assetPalette.refreshSiteAssets() 214
- assetPalette.removeFromFavorites(),
 - dreamweaver.assetPalette.removeFromFavorites() 214
- assetPalette.renameNickname(),
 - dreamweaver.assetPalette.renameNickname() 214
- assetPalette.setSelectedCategory(),
 - dreamweaver.assetPalette.setSelectedCategory() 215

- `assetPalette.setSelectedView()`
 - `dreamweaver.assetPalette.setSelectedView()` 215
- `attachExternalStylesheet()` 237
- `attributes` property 20



- `backspaceKey()` 310
- `balanceBracesTextView()` 375
- `beginReporting()` 57
- `behaviorFunction()` 92
- `behaviors`
 - API 89
 - helper functions 89
 - inserting multiple functions with 89
 - required functions 89
 - sample code 98
 - user experience 88
- `blur()` 14
- `body` property 19
- `boolean` object 14
- `bringDWToFront()` 101
- `bringFWToFront()` 102
- `browseDocument()` 243, 485
- `browseForFileURL()` 250
- `browseForFolderURL()` 251
- `button` object 14



- `C` functions
 - calling from JavaScript 202
 - in `mm_jsapi.h` 193
- `canAcceptBehavior()` 93
- `canAcceptCommand()`
 - in menu commands 47
 - in regular commands 40
- `canAddFrame()` 483
- `canAddKeyFrame()` 483
- `canAddLinkToFile()` 486
- `canAlign()` 465
- `canArrange()` 466
- `canChangeLink()` 486
- `canChangeObject()` 484
- `canCheckIn()` 487
- `canCheckOut()` 487
- `canClipCopy()` 476
- `canClipCopyText()` 466

- `canClipCut()` 477
- `canClipPaste()`
 - `dom.canClipPaste()` 467
 - `dreamweaver.canClipPaste` 477
- `canClipPasteText()` 467
- `canConnect()` 488
- `canConvertLayersToTable()` 467
- `canConvertTablesToLayers()` 468
- `canDecreaseColspan()` 468
- `canDecreaseRowspan()` 468
- `canDeleteTableColumn()` 469
- `canDeleteTableRow()` 469
- `canEditNoFramesContent()` 469
- `canEditSelection()` 483
- `canExportCSS()` 478
- `canFindLinkSource()` 488
- `canFindNext()` 478
- `canGet()` 488
- `canIncreaseColspan()` 470
- `canIncreaseRowspan()` 470
- `canInsertTableColumns()` 470
- `canInsertTableRows()` 471
- `canInspectSelection()` 72
- `canLocateInSite()` 489
- `canMakeEditable()` 489
- `canMakeNewEditableRegion()` 471
- `canMakeNewFileOrFolder()` 490
- `canMarkSelectionAsEditable()` 471
- `canMergeTableCells()` 472
- `canOpen()` 490
- `canOpenInFrame()` 478
- `canPlayPlugin()` 472
- `canPlayRecordedCommand()` 479
- `canPut()` 491
- `canRecreateCache()` 491
- `canRedo()`
 - `dom.canRedo()` 472
 - `dreamweaver.canRedo()` 479
- `canRefresh()` 492
- `canRemoveEditableRegion()` 473
- `canRemoveFrame()` 484
- `canRemoveKeyFrame()` 485
- `canRemoveLink()` 492
- `canRemoveObject()` 485
- `canRevertDocument()` 479
- `canSaveAll()` 480
- `canSaveDocument()` 480

- canSaveDocumentAsTemplate() 480
- canSaveFrameset() 481
- canSaveFramesetAs() 481
- canSelectAll() 481
- canSelectAllCheckedOutFiles() 493
- canSelectNewer() 493
- canSelectTable() 473
- canSetLayout() 492
- canSetLinkHref() 473
- canShowFindDialog() 482
- canShowListPropertiesDialog() 474
- canSplitFrame() 474
- canSplitTableCell() 474
- canStopPlugin() 475
- canSynchronize() 494
- canUndo()
 - dom.canUndo() 475
 - dreamweaver.canUndo() 482
- canUndoCheckOut() 494
- canViewAsRoot() 494
- changeLink() 350
- changeLinkSitewide() 350
- changeObject() 404
- checkbox object 14
- checkIn() 351
- checkLinks() 351
- checkOut() 352
- checkSpelling() 289
- checkTargetBrowsers()
 - dom.checkTargetBrowsers() 289
 - site.checkTargetBrowsers() 352
- childNodes property
 - of comment objects 22
 - of document objects 19
 - of tag objects 20
 - of text objects 22
- clearInterval() 14
- clearSteps() 295
- clearTemp() 138
- clearTimeout() 14
- clipCopy()
 - dom.clipCopy() 227
 - dreamweaver.clipCopy() 231
- clipCopyText() 228
- clipCut()
 - dom.clipCut() 228
 - dreamweaver.clipCut() 231
- clipPaste()
 - dom.clipPasteText() 229
 - dreamweaver.clipPaste() 232
- clipPasteText() 230
- close()
 - MMNotes.close() 116
 - window.close() 14
- closeDocument() 251
- CloseNotesFile() 121
- color button control 29
- columns 147
 - getting from stored procedures 153, 154
- commandButtons() 58
 - in menu commands 48
 - in regular commands 40
- commands
 - adding to menus 44
 - API 39
 - sample code 42
 - user experience 38
- comment object 22
- configureSettings() 58
- confirm() 14
- connections 150, 151, 152
 - getting list of 149
 - names 146
- convertLayersToTable() 234
- convertTablesToLayers() 235
- convertTo30() 235
- convertWidthsToPercent() 393
- convertWidthsToPixels() 394
- copy() 130
- copySteps() 295
- createDocument() 252
- createFile(), for Flash objects 110
- createFolder() 130
- createLayoutCell() 441
- createLayoutTable() 442
- createResultsWindow() 338
- Custom JavaScript Controls 24

D

- data property
 - of comment objects 22
 - of httpReply objects 137
 - of text objects 22
- date object 14
- debugDocument() 305
- decreaseColspan() 394
- decreaseRowspan() 394
- defineSites() 353
- deleteBehavior() 94
- deleteKey() 310
- deleteSelectedItem() 459
- deleteSelectedStyle()
 - dreamweaver.cssStylePalette.deleteSelectedStyle() 238
 - dreamweaver.htmlStylePalette.deleteSelectedStyle() 301
 - dw.htmlStylePalette.deleteSelectedStyle() 301
- deleteSelectedTemplate() 464
- deleteSelection()
 - dom.deleteSelection() 269
 - dreamweaver.deleteSelection() 284
 - site.deleteSelection() 353
- deleteTableColumn() 395
- deleteTableRow() 395
- Design Notes
 - C API 121
 - file structure 116
 - JavaScript API 116
 - user experience 116
- detachFromLibrary() 321
- detachFromTemplate() 321
- displayHelp()
 - in behavior actions 94
 - in floating panels 80
 - in object files 33
 - in Property Inspector files 73
- document node 19
- document object
 - DOM Level 1 properties and methods of 19
 - Netscape DOM properties and methods of 14
- document object model
 - DOM Level 1 specification 14
 - in Dreamweaver 14
- documentEdited() 80
- documentElement property 19
- doDeferredTableUpdate() 396
- doesColumnHaveSpacer() 442
- doesGroupHaveSpacers() 443
- DOM
 - DOM Level 1 specification 14
 - dom object 206
 - in Dreamweaver 14
- dom.addBehavior() 216
- dom.addSpacerToColumn() 441
- dom.align() 316
- dom.applyCharacterMarkup() 268
- dom.applyCSSStyle() 236
- dom.applyFontMarkup() 268
- dom.applyHTMLStyle() 300
- dom.applyTemplate() 320
- dom.arrange() 317
- dom.arrowDown() 308
- dom.arrowLeft() 308
- dom.arrowRight() 309
- dom.arrowUp() 309
- dom.backspaceKey() 310
- dom.canAlign() 465
- dom.canArrange() 466
- dom.canClipCopyText() 466
- dom.canClipPaste() 467
- dom.canClipPasteText() 467
- dom.canConvertLayersToTable() 467
- dom.canConvertTablesToLayers() 468
- dom.canDecreaseColspan() 468
- dom.canDecreaseRowspan() 468
- dom.canDeleteTableColumn() 469
- dom.canDeleteTableRow() 469
- dom.canEditNoFramesContent() 469
- dom.canIncreaseColspan() 470
- dom.canIncreaseRowspan() 470
- dom.canInsertTableColumns() 470
- dom.canInsertTableRows() 471
- dom.canMakeNewEditableRegion() 471
- dom.canMarkSelectionAsEditable() 471
- dom.canMergeTableCells() 472
- dom.canPlayPlugin() 472
- dom.canRedo() 472
- dom.canRemoveEditableRegion() 473
- dom.canSelectTable() 473
- dom.canSetLinkHref() 473
- dom.canShowListPropertiesDialog() 474
- dom.canSplitFrame() 474

- dom.canSplitTableCell() 474
- dom.canStopPlugin() 475
- dom.canUndo() 475
- dom.checkSpelling() 289
- dom.checkTargetBrowsers() 289
- dom.clipCopy 227
- dom.clipCopyText() 228
- dom.clipCut() 228
- dom.clipPaste 229
- dom.clipPasteText() 230
- dom.convertLayersToTable() 234
- dom.convertTablesToLayers() 235
- dom.convertTo30() 235
- dom.convertWidthsToPercent() 393
- dom.convertWidthsToPixels() 394
- dom.createLayoutCell() 441
- dom.createLayoutTable() 442
- dom.decreaseColspan() 394
- dom.decreaseRowspan() 394
- dom.deleteKey() 310
- dom.deleteSelection() 269
- dom.deleteTableColumn() 395
- dom.deleteTableRow() 395
- dom.detachFromLibrary() 321
- dom.detachFromTemplate() 321
- dom.doDeferredTableUpdate() 396
- dom.doesColumnHaveSpacer() 442
- dom.doesGroupHaveSpacers() 443
- dom.editAttribute() 269
- dom.endOfDocument() 311
- dom.endOfLine() 311
- dom.exitBlock() 270
- dom.getAttachedTemplate() 322
- dom.getBehavior() 217
- dom.getCharSet() 270
- dom.getClickedHeaderColumn() 443
- dom.getEditableRegionList() 322
- dom.getEditableRetionList() 324
- dom.getEditNoFramesContent() 409, 410
- dom.getFocus() 447
- dom.getFontMarkup() 270
- dom.getFrameNames() 266
- dom.getIsLibraryDocument() 323
- dom.getIsTemplateDocument() 323
- dom.getLinkHref() 271
- dom.getLinkTarget() 271
- dom.getListTag() 272
- dom.getPreventLayerOverlaps() 410
- dom.getRulerOrigin() 434
- dom.getRulerUnits() 434
- dom.getSelectedEditableRegion() 324
- dom.getSelectedNode() 341
- dom.getSelection() 342
- dom.getShowAutoIndent() 410
- dom.getShowFrameBorders() 411
- dom.getShowGrid() 411
- dom.getShowHeaderView() 412
- dom.getShowHighlightInvalidHTML() 412
- dom.getShowImageMaps() 412
- dom.getShowInvisibleElements() 413
- dom.getShowLayerBorders() 413, 421
- dom.getShowLayoutTableTabs() 443
- dom.getShowLayoutView() 444
- dom.getShowLineNumbers() 413
- dom.getShowRulers() 414
- dom.getShowSyntaxColoring() 414
- dom.getShowTableBorders() 414
- dom.getShowToolbar() 415
- dom.getShowTracingImage() 415
- dom.getShowWordWrap() 415
- dom.getSnapToGrid() 416
- dom.getTableExtent() 396
- dom.getTextAlignment() 272
- dom.getTextFormat() 273
- dom.getTracingImageOpacity() 435
- dom.getView() 448
- dom.getWindowTitle() 448
- dom.hasCharacterMarkup() 273
- dom.hasTracingImage() 475
- dom.increaseColspan() 397
- dom.increaseRowspan() 397
- dom.indent() 274
- dom.insertHTML() 274
- dom.insertLibraryItem() 324
- dom.insertObject() 275
- dom.insertTableColumns() 398
- dom.insertTableRows() 398
- dom.insertText() 276
- dom.instrumentDocument() 304
- dom.isColumnAutostretch() 444
- dom.isDesignViewUpdated() 372
- dom.isDocumentInFrame() 266
- dom.isSelectionValid() 373
- dom.loadTracingImage() 435

dom.makeCellWidthsConsistent() 444
 dom.makeSizesEqual() 317
 dom.markSelectionAsEditable() 325
 dom.mergeTableCells() 399
 dom.moveSelectionBy() 318
 dom.newBlock() 277
 dom.newEditableRegion() 325
 dom.nextParagraph() 312
 dom.nextWord() 312
 dom.nodeToOffsets() 343
 dom.notifyFlashObjectChanged() 277
 dom.offsetsToNode() 344
 dom.outdent() 278
 dom.pageDown() 313
 dom.pageUp() 313
 dom.playAllPlugins() 435
 dom.playPlugin() 436
 dom.previousParagraph() 314
 dom.previousWord() 314
 dom.reapplyBehaviors() 218
 dom.redo() 291
 dom.removeAllSpacers() 445
 dom.removeAllTableHeights() 399
 dom.removeAllTableWidths() 399
 dom.removeBehavior() 218
 dom.removeCharacterMarkup() 278
 dom.removeCSSStyle() 237
 dom.removeEditableRegion() 326
 dom.removeFontMarkup() 279
 dom.removeLink() 279
 dom.removeSpacerFromColumn() 445
 dom.resizeSelection() 280
 dom.resizeSelectionBy() 318
 dom.runTranslator() 431
 dom.saveAllFrames() 267
 dom.selectAll() 345
 dom.selectChild() 333
 dom.selectParent() 334
 dom.selectTable() 345
 dom.setAttributeWithErrorChecking() 280
 dom.setColumnAutostretch() 446
 dom.setEditNoFramesContent() 416
 dom.setHideAllVisualAids() 417
 dom.setLayerTag() 319
 dom.setLinkHref() 281
 dom.setLinkTarget() 281
 dom.setListBoxKind() 282
 dom.setListTag() 282
 dom.setPreventLayerOverlaps() 417
 dom.setRulerOrigin() 436
 dom.setRulerUnits() 437
 dom.setSelectedNode() 346
 dom.setSelection() 347
 dom.setShowFrameBorders() 418
 dom.setShowGrid() 418
 dom.setShowHeaderView() 418
 dom.setShowHighlightInvalidHTML() 419
 dom.setShowImageMaps() 419
 dom.setShowInvisibleElements() 419
 dom.setShowLayerBorders() 420
 dom.setShowLayoutTableTabs() 446
 dom.setShowLayoutView() 447
 dom.setShowLineNumbers() 420
 dom.setShowRulers() 420
 dom.setShowTableBorders() 421
 dom.setShowToolbar() 421
 dom.setShowTracingImage() 422
 dom.setShowWordWrap() 422
 dom.setSnapToGrid() 422
 dom.setTableCellTag() 400
 dom.setTableColumns() 400
 dom.setTableRows() 401
 dom.setTextAlignment() 283
 dom.setTextFieldKind() 283
 dom.setTracingImageOpacity() 438
 dom.setTracingImagePosition() 437
 dom.setView() 448
 dom.showFontColorDialog() 284
 dom.showInsertTableRowsOrColumnsDialog() 401
 dom.showListPropertiesDialog() 282
 dom.showPagePropertiesDialog() 290
 dom.snapTracingImageToSelection() 438
 dom.source.arrowDown() 373
 dom.source.arrowLeft() 374
 dom.source.arrowRight() 374
 dom.source.arrowUp() 375
 dom.source.balanceBracesTextView() 375
 dom.source.endOfDocument() 376
 dom.source.endOfLine() 376
 dom.source.endPage() 377
 dom.source.getCurrentLines() 377
 dom.source.getSelection() 378
 dom.source.getText() 378
 dom.source.indentTextView() 378

- dom.source.insert() 379
- dom.source.nextWord() 379
- dom.source.outdentTextView() 380
- dom.source.pageDown() 380
- dom.source.pageUp() 381
- dom.source.previousWord() 381
- dom.source.replaceRange() 382
- dom.source.scrollLineDown() 382, 383, 384
- dom.source.scrollLineUp() 383
- dom.source.scrollPageDown() 383
- dom.source.scrollPageUp() 384
- dom.source.selectParentTag() 384
- dom.source.setCurrentLine() 385
- dom.source.startOfDocument() 385
- dom.source.startOfLine() 386
- dom.source.synchronizeDocument() 387
- dom.source.topPage() 386
- dom.source.wrapSelection() 387
- dom.splitFrame() 267
- dom.splitTableCell() 402
- dom.startOfDocument() 315
- dom.startOfLine() 315
- dom.stopAllPlugins() 439
- dom.stopPlugin() 439
- dom.stripTag() 334
- dom.undo() 292
- dom.updateCurrentPage() 326
- dom.wrapTag() 335
- doURLEncoding() 388
- dreamweaver object
 - methods of 206
 - properties of 18
- dreamweaver.arrangeFloatingPalettes() 440
- dreamweaver.behaviorInspector object 216
- dreamweaver.behaviorInspector.getBehaviorAt() 222
- dreamweaver.behaviorInspector.getBehaviorCount() 222
- dreamweaver.behaviorInspector.getSelectedBehavior() 223
- dreamweaver.behaviorInspector.moveBehaviorDown() 224
- dreamweaver.behaviorInspector.moveBehaviorUp() 225
- dreamweaver.behaviorInspector.setSelectedBehavior() 226
- dreamweaver.browseDocument() 243
- dreamweaver.browseForFileURL() 250
- dreamweaver.browseForFolderURL() 251
- dreamweaver.canClipCopy() 476
- dreamweaver.canClipCut() 477
- dreamweaver.canClipPaste() 477
- dreamweaver.canExportCSS() 478
- dreamweaver.canFindNext() 478
- dreamweaver.canOpenInFrame() 478
- dreamweaver.canPlayRecordedCommand() 479
- dreamweaver.canRedo() 479
- dreamweaver.canRevertDocument() 479
- dreamweaver.canSaveAll() 480
- dreamweaver.canSaveDocument() 480
- dreamweaver.canSaveDocumentAsTemplate() 480
- dreamweaver.canSaveFrameset() 481
- dreamweaver.canSaveFramesetAs() 481
- dreamweaver.canSelectAll() 481
- dreamweaver.canShowFindDialog() 482
- dreamweaver.canUndo() 482
- dreamweaver.clipCopy() 231
- dreamweaver.clipCut() 231
- dreamweaver.clipPaste() 232
- dreamweaver.closeDocument() 251
- dreamweaver.createDocument() 252
- dreamweaver.createResultsWindow() 338
- dreamweaver.cssStylePalette object 236
- dreamweaver.cssStylePalette.deleteSelectedStyle() 238
- dreamweaver.cssStylePalette.duplicateSelectedStyle() 238
- dreamweaver.cssStylePalette.editSelectedStyle() 239
- dreamweaver.cssStylePalette.editStyleSheet() 239
- dreamweaver.cssStylePalette.getSelectedStyle() 240
- dreamweaver.cssStylePalette.getSelectedTarget() 241
- dreamweaver.cssStylePalette.getStyles() 242
- dreamweaver.cssStylePalette.newStyle() 243
- dreamweaver.debugDocument() 305
- dreamweaver.deleteSelection() 284
- dreamweaver.doURLEncoding() 388
- dreamweaver.editCommandList() 233
- dreamweaver.editFontList() 284
- dreamweaver.editLockedRegions() 432
- dreamweaver.exportCSS() 252
- dreamweaver.exportEditableRegionsAsXML() 253
- dreamweaver.findNext() 260
- dreamweaver.getActiveWindow() 449
- dreamweaver.getBehaviorElement() 219
- dreamweaver.getBehaviorEvent() 456
- dreamweaver.getBehaviorTag() 220
- dreamweaver.getBrowserList() 244
- dreamweaver.getClipboardText() 232
- dreamweaver.getConfigurationPath() 330
- dreamweaver.getDebugBrowserList() 305
- dreamweaver.getDocumentDOM() 207
- dreamweaver.getDocumentList() 449

- dreamweaver.getDocumentPath() 331
- dreamweaver.getElementRef() 290
- dreamweaver.getExtensionEditorList() 245
- dreamweaver.getExternalTextEditor() 245
- dreamweaver.getFloaterVisibility() 450
- dreamweaver.getFocus() 450
- dreamweaver.getFontList() 285
- dreamweaver.getFontStyles() 285
- dreamweaver.getHideAllFloaters() 423
- dreamweaver.getIsAnyBreakpoints() 306
- dreamweaver.getKeyState() 286
- dreamweaver.getMenuNeedsUpdating() 328
- dreamweaver.getObjectRefs() 456
- dreamweaver.getObjectTags() 458
- dreamweaver.getPrimaryBrowser() 246
- dreamweaver.getPrimaryExtensionEditor() 246
- dreamweaver.getPrimaryView() 451
- dreamweaver.getRecentFileList() 253
- dreamweaver.getRedoText() 292
- dreamweaver.getSecondaryBrowser() 247
- dreamweaver.getShowDialogsOnInsert() 287
- dreamweaver.getShowStatusBar() 423
- dreamweaver.getSiteRoot() 332
- dreamweaver.getSnapDistance() 451
- dreamweaver.getSystemFontList() 286
- dreamweaver.getTokens() 389
- dreamweaver.getTranslatorList() 432
- dreamweaver.getUndoText() 292
- dreamweaver.historyPalette object 291
- dreamweaver.historyPalette.clearSteps() 295
- dreamweaver.historyPalette.copySteps() 295
- dreamweaver.historyPalette.getSelectedSteps() 296
- dreamweaver.historyPalette.getStepCount() 296
- dreamweaver.historyPalette.getStepsAsJavaScript() 297
- dreamweaver.historyPalette.getUndoState() 298
- dreamweaver.historyPalette.replaySteps() 298
- dreamweaver.historyPalette.saveAsCommand() 299
- dreamweaver.historyPalette.setSelectedSteps() 299
- dreamweaver.historyPalette.setUndoState() 300
- dreamweaver.htmlInspector.getShowAutoIndent() 423
- dreamweaver.htmlInspector.getShowHighlightInvalid HTML() 424
- dreamweaver.htmlInspector.getShowLineNumbers() 424
- dreamweaver.htmlInspector.getShowSyntaxColoring() 424
- dreamweaver.htmlInspector.getShowWordWrap() 425
- dreamweaver.htmlInspector.setShowAutoIndent() 425
- dreamweaver.htmlInspector.setShowHighlightInvalid HTML() 425
- dreamweaver.htmlInspector.setShowLineNumbers() 426
- dreamweaver.htmlInspector.setShowSyntaxColoring() 426
- dreamweaver.htmlInspector.setShowWordWrap() 426
- dreamweaver.htmlStylePalette object 300
- dreamweaver.htmlStylePalette.canEditSelection() 483
- dreamweaver.htmlStylePalette.deleteSelectedStyle() 301
- dreamweaver.htmlStylePalette.duplicateSelectedStyle() 301
- dreamweaver.htmlStylePalette.editSelectedStyle() 301
- dreamweaver.htmlStylePalette.getSelectedStyle() 302
- dreamweaver.htmlStylePalette.getStyles() 302
- dreamweaver.htmlStylePalette.newStyle() 303
- dreamweaver.htmlStylePalette.setSelectedStyle() 303
- dreamweaver.importXMLIntoTemplate() 253
- dreamweaver.isRecording() 482
- dreamweaver.isReporting() 336
- dreamweaver.latin1ToNative() 390
- dreamweaver.libraryPalette object 320
- dreamweaver.libraryPalette.deleteSelectedItem() 459
- dreamweaver.libraryPalette.getSelectedItem() 460
- dreamweaver.libraryPalette.newFromDocument() 460
- dreamweaver.libraryPalette.recreateFromDocument() 461
- dreamweaver.libraryPalette.renameSelectedItem() 461
- dreamweaver.loadSitesFromPrefs() 348
- dreamweaver.minimizeRestoreAll() 452
- dreamweaver.nativeToLatin1() 390
- dreamweaver.newFromTemplate() 254
- dreamweaver.nodeToOffsets() 462
- dreamweaver.notifyMenuUpdated() 329
- dreamweaver.offsetsToNode() 463
- dreamweaver.openDocument() 254
- dreamweaver.openDocumentFromSite() 255
- dreamweaver.openInFrame() 255
- dreamweaver.openWithApp() 247
- dreamweaver.openWithBrowseDialog() 248
- dreamweaver.openWithExternalTextEditor() 248
- dreamweaver.openWithImageEditor() 249
- dreamweaver.playRecordedCommand() 293
- dreamweaver.popupAction() 221
- dreamweaver.popupCommand() 463
- dreamweaver.quitApplication() 287
- dreamweaver.redo() 293
- dreamweaver.relativeToAbsoluteURL() 332
- dreamweaver.releaseDocument() 256
- dreamweaver.reloadMenus() 329
- dreamweaver.removeAllBreakpoints() 306

- dreamweaver.replace() 260
- dreamweaver.replaceAll() 261
- dreamweaver.results.setResultData() 337
- dreamweaver.revertDocument() 256
- dreamweaver.runCommand() 233
- dreamweaver.saveAll() 257
- dreamweaver.saveDocument() 257
- dreamweaver.saveDocumentAs() 258
- dreamweaver.saveDocumentAsTemplate() 258
- dreamweaver.saveFrameset() 259
- dreamweaver.saveFramesetAs() 259
- dreamweaver.saveSitesToPrefs() 348
- dreamweaver.selectAll() 347
- dreamweaver.setActiveWindow() 452
- dreamweaver.setFloaterVisibility() 453
- dreamweaver.setHideAllFloaters() 427
- dreamweaver.setPrimaryView() 453
- dreamweaver.setSelection() 464
- dreamweaver.setShowStatusBar() 427
- dreamweaver.setSnapDistance() 454
- dreamweaver.setUpComplexFind() 261
- dreamweaver.setUpComplexFindReplace() 262
- dreamweaver.setUpFind() 263
- dreamweaver.setUpFindReplace() 264
- dreamweaver.showAboutBox() 288
- dreamweaver.showFindDialog() 265
- dreamweaver.showFindReplaceDialog() 265
- dreamweaver.showGridSettingsDialog() 440
- dreamweaver.showPreferencesDialog() 288
- dreamweaver.showProperties() 454
- dreamweaver.showQuickTagEditor() 335
- dreamweaver.showReportsDialog() 336
- dreamweaver.startDebugger() 307
- dreamweaver.startRecording() 294
- dreamweaver.stopRecording() 294
- dreamweaver.stylePalette.attachExternalStylesheet() 237
- dreamweaver.templatePalette object 320
- dreamweaver.templatePalette.deleteSelectedTemplate() 464
- dreamweaver.templatePalette.getSelectedTemplate() 462
- dreamweaver.templatePalette.newBlankTemplate() 327
- dreamweaver.templatePalette.renameSelected
Template() 465
- dreamweaver.timelineInspector object 402
- dreamweaver.timelineInspector.addBehavior() 402
- dreamweaver.timelineInspector.addFrame() 403
- dreamweaver.timelineInspector.addKeyframe() 403
- dreamweaver.timelineInspector.addObject() 403

- dreamweaver.timelineInspector.addTimeline() 404
- dreamweaver.timelineInspector.canAddFrame() 483
- dreamweaver.timelineInspector.canAddKeyFrame() 483
- dreamweaver.timelineInspector.canChangeObject() 484
- dreamweaver.timelineInspector.canRemoveFrame() 484
- dreamweaver.timelineInspector.canRemoveKey
Frame() 485
- dreamweaver.timelineInspector.canRemoveObject() 485
- dreamweaver.timelineInspector.changeObject() 404
- dreamweaver.timelineInspector.getAutoplay() 404
- dreamweaver.timelineInspector.getCurrentFrame() 405
- dreamweaver.timelineInspector.getLoop() 405
- dreamweaver.timelineInspector.recordPathOfLayer() 405
- dreamweaver.timelineInspector.removeBehavior() 406
- dreamweaver.timelineInspector.removeFrame() 406
- dreamweaver.timelineInspector.removeKeyframe() 406
- dreamweaver.timelineInspector.removeObject() 407
- dreamweaver.timelineInspector.removeTimeline() 407
- dreamweaver.timelineInspector.renameTimeline() 407
- dreamweaver.timelineInspector.setAutoplay() 408
- dreamweaver.timelineInspector.setCurrentFrame() 408
- dreamweaver.timelineInspector.setLoop() 409
- dreamweaver.toggleFloater() 455
- dreamweaver.undo() 294
- dreamweaver.updatePages() 327
- dreamweaver.updateReference() 455
- dreamweaver.useTranslatedSource() 433
- drivers 151
- duplicateSelectedStyle()
 - dreamweaver.cssStylePalette.duplicateSelected
Style() 238
 - dreamweaver.htmlStylePalette.duplicateSelected
Style() 301
- dw object 206
- dw.arrangeFloatingPalettes() 440
- dw.assetPalette.addToFavoritesFromDocument() 208
- dw.assetPalette.addToFavoritesFromSiteAssets() 208
- dw.assetPalette.addToFavoritesFromSiteWindow() 209
- dw.assetPalette.canEdit() 476
- dw.assetPalette.canInsertOrApply() 476
- dw.assetPalette.copyToSite() 209
- dw.assetPalette.edit() 210
- dw.assetPalette.getSelectedCategory() 210
- dw.assetPalette.getSelectedItems() 211
- dw.assetPalette.getSelectedView() 211
- dw.assetPalette.insertOrApply() 212
- dw.assetPalette.locateInSite() 212

dw.assetPalette.newAsset() 213
 dw.assetPalette.newFolder() 213
 dw.assetPalette.recreateLibraryFromDocument() 213
 dw.assetPalette.refreshSiteAssets() 214
 dw.assetPalette.removeFromFavorites() 214
 dw.assetPalette.renameNickname() 214
 dw.assetPalette.setSelectedCategory() 215
 dw.assetPalette.setSelectedView() 215
 dw.behaviorInspector.getBehaviorAt() 222
 dw.behaviorInspector.getBehaviorCount() 222
 dw.behaviorInspector.getSelectedBehavior() 223
 dw.behaviorInspector.moveBehaviorDown() 224
 dw.behaviorInspector.moveBehaviorUp() 225
 dw.behaviorInspector.setSelectedBehavior() 226
 dw.browseDocument() 243
 dw.browseForFileURL() 250
 dw.browseForFolderURL() 251
 dw.canClipCopy() 476
 dw.canClipCut() 477
 dw.canClipPaste() 477
 dw.canExportCSS() 478
 dw.canFindNext() 478
 dw.canOpenInFrame() 478
 dw.canPlayRecordedCommand() 479
 dw.canRedo() 479
 dw.canRevertDocument() 479
 dw.canSaveAll() 480
 dw.canSaveDocument() 480
 dw.canSaveDocumentAsTemplate() 480
 dw.canSaveFrameset() 481
 dw.canSaveFramesetAs() 481
 dw.canSelectAll() 481
 dw.canShowFindDialog() 482
 dw.canUndo() 482
 dw.clipCopy() 231
 dw.clipCut() 231
 dw.clipPaste() 232
 dw.closeDocument() 251
 dw.createDocument() 252
 dw.createResultsWindow() 338
 dw.cssStylePalette.deleteSelectedStyle() 238
 dw.cssStylePalette.duplicateSelectedStyle() 238
 dw.cssStylePalette.editSelectedStyle() 239
 dw.cssStylePalette.editStyleSheet() 239
 dw.cssStylePalette.getSelectedStyle() 240
 dw.cssStylePalette.getSelectedTarget() 241
 dw.cssStylePalette.getStyles() 242
 dw.cssStylePalette.newStyle() 243
 dw.debugDocument() 305
 dw.deleteSelection() 284
 dw.doURLEncoding() 388
 dw.editCommandList() 233
 dw.editFontList() 284
 dw.editLockedRegions() 432
 dw.exportCSS() 252
 dw.exportEditableRegionsAsXML() 253
 DWfile DLL
 API 130
 checking for 129
 DWfile.copy() 130
 DWfile.createFolder() 130
 DWfile.exists() 131
 DWfile.getAttributes() 132
 DWfile.getCreationDate() 134
 DWfile.getModificationDate() 133
 DWfile.listFolder() 134
 DWfile.read() 135
 DWfile.remove() 136
 DWfile.write() 136
 dw.findNext() 260
 dw.getActiveWindow() 449
 dw.getBehaviorElement() 219
 dw.getBehaviorTag() 220
 dw.getBrowserList() 244
 dw.getConfigurationPath() 330
 dw.getDebugBrowserList() 305
 dw.getDocumentDOM() 207
 dw.getDocumentList() 449
 dw.getDocumentPath() 331
 dw.getElementRef() 290
 dw.getExtensionEditorList() 245
 dw.getExternalTextEditor() 245
 dw.getFloaterVisibility() 450
 dw.getFocus() 450
 dw.getFontList() 285
 dw.getFontStyles() 285
 dw.getHideAllFloaters() 423
 dw.getIsAnyBreakpoints() 306
 dw.getKeyState() 286
 dw.getMenuNeedsUpdating() 328
 dw.getPrimaryBrowser() 246
 dw.getPrimaryExtensionEditor() 246
 dw.getPrimaryView() 451
 dw.getRecentFileList() 253

dw.getRedoText() 292
 dw.getSecondaryBrowser() 247
 dw.getShowDialogsOnInsert() 287
 dw.getShowStatusBar() 423
 dw.getSiteRoot() 332
 dw.getSnapDistance() 451
 dw.getSystemFontList() 286
 dw.getTokens() 389
 dw.getTranslatorList() 432
 dw.getUndoText() 292
 dw.historyPalette.clearSteps() 295
 dw.historyPalette.copySteps() 295
 dw.historyPalette.getSelectedSteps() 296
 dw.historyPalette.getStepCount() 296
 dw.historyPalette.getStepsAsJavaScript() 297
 dw.historyPalette.getUndoState() 298
 dw.historyPalette.replaySteps() 298
 dw.historyPalette.saveAsCommand() 299
 dw.historyPalette.setSelectedSteps() 299
 dw.historyPalette.setUndoState() 300
 dw.htmlInspector.getShowAutoIndent() 423
 dw.htmlInspector.getShowHighlightInvalid
 HTML() 424
 dw.htmlInspector.getShowLineNumbers() 424
 dw.htmlInspector.getShowSyntaxColoring() 424
 dw.htmlInspector.getShowWordWrap() 425
 dw.htmlInspector.setShowAutoIndent() 425
 dw.htmlInspector.setShowHighlightInvalid
 HTML() 425
 dw.htmlInspector.setShowLineNumbers() 426
 dw.htmlInspector.setShowSyntaxColoring() 426
 dw.htmlInspector.setShowWordWrap() 426
 dw.htmlStylePalette.canEditSelection() 483
 dw.htmlStylePalette.deleteSelectedStyle() 301
 dw.htmlStylePalette.duplicateSelectedStyle() 301
 dw.htmlStylePalette.editSelectedStyle() 301
 dw.htmlStylePalette.getSelectedStyle() 302
 dw.htmlStylePalette.getStyles() 302
 dw.htmlStylePalette.newStyle() 303
 dw.htmlStylePalette.setSelectedStyle() 303
 dw.importXMLIntoTemplate() 253
 dw.isRecording() 482
 dw.isReporting() 336
 dw.latin1ToNative() 390
 dw.libraryPalette.deleteSelectedItem() 459
 dw.libraryPalette.getSelectedItem() 460
 dw.libraryPalette.newFromDocument() 460
 dw.libraryPalette.recreateFromDocument() 461
 dw.libraryPalette.renameSelectedItem() 461
 dw.loadSitesFromPrefs() 348
 dw.minimizeRestoreAll() 452
 dw.nativeToLatin1() 390
 dw.newFromTemplate() 254
 dw.notifyMenuUpdated() 329
 dw.openDocument() 254
 dw.openDocumentFromSite() 255
 dw.openInFrame() 255
 dw.openWithApp() 247
 dw.openWithBrowseDialog() 248
 dw.openWithExternalTextEditor() 248
 dw.openWithImageEditor() 249
 dw.playRecordedCommand() 293
 dw.popupAction() 221
 dw.quitApplication() 287
 dw.redo() 293
 dw.referencePalette.getFontSize() 215
 dw.referencePalette.setFontSize() 216
 dw.relativeToAbsoluteURL() 332
 dw.releaseDocument() 256
 dw.reloadMenus() 329
 dw.removeAllBreakpoints() 306
 dw.replace() 260
 dw.replaceAll() 261
 dw.results.setResultData() 337
 dw.revertDocument() 256
 dw.runCommand() 233
 dw.saveAll() 257
 dw.saveDocument() 257
 dw.saveDocumentAs() 258
 dw.saveDocumentAsTemplate() 258
 dw.saveFrameset() 259
 dw.saveFramesetAs() 259
 dw.saveSitesToPrefs() 348
 dw.selectAll() 347
 dw.setActiveWindow() 452
 dw.setFloaterVisibility() 453
 dw.setHideAllFloaters() 427
 dw.setPrimaryView() 453
 dw.setShowStatusBar() 427
 dw.setSnapDistance() 454
 dw.setUpComplexFind() 261
 dw.setUpComplexFindReplace() 262
 dw.setUpFind() 263
 dw.setUpFindReplace() 264

- dw.showAboutBox() 288
- dw.showFindDialog() 265
- dw.showFindReplaceDialog() 265
- dw.showGridSettingsDialog() 440
- dw.showPreferencesDialog() 288
- dw.showProperties() 454
- dw.showQuickTagEditor() 335
- dw.showReportsDialog() 336
- dw.startDebugger() 307
- dw.startRecording() 294
- dw.stopRecording() 294
- dw.stylePalette.attachExternalStylesheet() 237
- dw.templatePalette.deleteSelectedTemplate() 464
- dw.templatePalette.getSelectedTemplate() 462
- dw.templatePalette.newBlankTemplate() 327
- dw.templatePalette.renameSelectedTemplate() 465
- dw.timelineInspector.addBehavior() 402
- dw.timelineInspector.addFrame() 403
- dw.timelineInspector.addKeyframe() 403
- dw.timelineInspector.addObject() 403
- dw.timelineInspector.addTimeline() 404
- dw.timelineInspector.canAddFrame() 483
- dw.timelineInspector.canAddKeyFrame() 483
- dw.timelineInspector.canChangeObject() 484
- dw.timelineInspector.canRemoveFrame() 484
- dw.timelineInspector.canRemoveKeyFrame() 485
- dw.timelineInspector.canRemoveObject() 485
- dw.timelineInspector.changeObject() 404
- dw.timelineInspector.getAutoplay() 404
- dw.timelineInspector.getCurrentFrame() 405
- dw.timelineInspector.getLoop() 405
- dw.timelineInspector.recordPathOfLayer() 405
- dw.timelineInspector.removeBehavior() 406
- dw.timelineInspector.removeFrame() 406
- dw.timelineInspector.removeKeyframe() 406
- dw.timelineInspector.removeObject() 407
- dw.timelineInspector.removeTimeline() 407
- dw.timelineInspector.renameTimeline() 407
- dw.timelineInspector.setAutoplay() 408
- dw.timelineInspector.setCurrentFrame() 408
- dw.timelineInspector.setLoop() 409
- dw.toggleFloater() 455
- dw.undo() 294
- dw.updatePages() 327
- dw.updateReference() 455
- dw.useTranslatedSource() 433

- dynamic menus
 - sample code 53
 - user experience 46



- editAttribute() 269
- editColumns(), site.editColumns() 353
- editCommandList() 233
- editFontList() 284
- editLockedRegions() 432
- editSelectedStyle()
 - dreamweaver.cssStylePalette.editSelectedStyle() 239
 - dreamweaver.htmlStylePalette.editSelectedStyle() 301
- editStyleSheet() 239
- element node 20
- enablers
 - return value 465
 - using 206
- endOfDocument() 311, 376
- endOfLine() 311, 376
- endPage() 377
- endReporting() 58
- escape() 14
- event handlers
 - in behavior dialog boxes 88
 - in extension files 23
 - returning a value from 89
- events, in extension files 14
- execJsInFireworks() 102
- exists() 131
- exitBlock() 270
- exportCSS() 252
- exportEditableRegionsAsXML() 253
- external JavaScript files 23



- file (field) object 14
- file i/o 130
- FilePathToLocalURL() 122
- filePathToLocalURL() 117
- files on disk
 - copying 130
 - creating (HTML files) 252
 - creating (non-HTML files) 136
 - reading 135
 - removing 136
 - writing to 136
- findLinkSource() 354
- findNext() 260
- Fireworks, integration example 106
- Flash objects, creating 110
- floating panels
 - API 79
 - performance issues 83
 - sample code 85
 - user experience 78
- focus() 14
- form object 14
- function object 14
- FWLaunch.bringDWToFront() 101
- FWLaunch.bringFWToFront() 102
- FWLaunch.execJsInFireworks() 102
- FWLaunch.getJsResponse() 103
- FWLaunch.mayLaunchFireworks() 104
- FWLaunch.optimizeInFireworks() 105
- FWLaunch.validateFireworks() 106



- get()
 - MMNotes.get() 117
 - site.get() 355
- getActiveWindow() 449
- getAttachedTemplate() 322
- getAttribute() 20
- getAttributes() 132
- getAutoplay() 404
- getBehavior() 217
- getBehaviorAt() 222
- getBehaviorCount() 222
- getBehaviorElement() 219
- getBehaviorEvent() 456
- getBehaviorTag() 220

- getBodyInstrument() 65
- getBrowserList() 244
- getCharSet() 270
- getCheckOutUser() 355
- getCheckOutUserForFile() 356
- getClickedHeaderColumn() 443
- getClipboardText(), dreamweaver.getClipboardText() 232
- getColdFusionDsnList() 147
- getColumnAndTypeList() 147
- getColumnList() 148
- getColumnsOfTable() 149
- getConfigurationPath() 330
- getConnectionList() 149
- getConnectionName() 150
- getConnectionState() 356
- getConnectionString() 151
- getCurrentFrame() 405
- getCurrentLines() 377
- getCurrentSite() 357
- getDate() 134
- getDebugBrowserList() 305
- getDocumentDOM()
 - dreamweaver.getDocumentDOM() 207
 - importance of 207
- getDocumentList() 449
- getDocumentPath() 331
- getDriverName() 151
- getDynamicContent() 48
- getEditableRegionList() 322
- getEditableRetionList() 324
- getEditNoFramesContent() 409, 410
- getElementRef() 290
- getElementsByTagName()
 - for document objects 19
 - for tag objects 20
- getExtensionEditorList() 245
- getExternalTextEditor() 245
- getFile() 138
- getFileCallback() 140
- getFloaterVisibility() 450
- getFocus()
 - dom.getFocus() 447
 - dreamweaver.getFocus() 450
 - site.getFocus() 357
- getFontList() 285
- getFontMarkup() 270
- getFontStyles() 285

- getFrameNames() 266
- getFunctionEndInstrument() 64
- getFunctionStartInstrument() 65
- getHeadInstrument() 65
- getHideAllFloaters() 423
- getIncludedFileList() 66
- getIsAnyBreakpoints() 306
- getIsLibraryDocument() 323
- getIsTemplateDocument() 323
- getJsResponse() 103
- getKeyCount() 117
- getKeys() 118
- getKeyState() 286
- getLinkHref() 271
- getLinkTarget() 271
- getLinkVisibility() 358
- getListTag() 272
- getLoop() 405
- getMenuNeedsUpdating() 328
- getModificationDate() 133
- getNaturalSize(), for Flash objects 111
- GetNote() 122
- GetNoteLength() 123
- GetNotesKeyCount() 123
- GetNotesKeys() 124
- getObjectRefs() 456
- getObjectTags() 458
- getObjectType(), for Flash objects 112
- getOnUnloadInstrument() 66
- getPassword() 152
- getPreventLayerOverlaps() 410
- getPrimaryBrowser() 246
- getPrimaryExtensionEditor() 246
- getPrimaryView() 451
- getRecentFileList() 253
- getRedoText() 292
- getRulerOrigin() 434
- getRulerUnits() 434
- getRuntimeConnectionType() 152
- getSecondaryBrowser() 247
- getSelectedBehavior() 223
- getSelectedEditableRegion() 324
- getSelectedItem() 460
- getSelectedNode() 341
- getSelectedStyle()
 - dreamweaver.cssStylePalette.getSelectedStyle() 240
 - dreamweaver.htmlStylePalette.getSelectedStyle() 302
- getSelectedTarget() 241
- getSelectedTemplate() 462
- getSelection() 378
 - dom.getSelection() 342
 - dreamweaver.getSelection() 459
 - site.getSelection() 358
- getShowAutoIndent() 410
- getShowDependents() 428
- getShowDialogsOnInsert() 287
- getShowFrameBorders() 411
- getShowGrid() 411
- getShowHeaderView() 412
- getShowHiddenFiles() 428
- getShowHighlightInvalidHTML() 412
- getShowImageMaps() 412
- getShowInvisibleElements() 413
- getShowLayerBorders() 413, 421
- getShowLayoutTableTabs() 443
- getShowLayoutView() 444
- getShowLineNumbers() 413
- getShowPageTitles() 429
- getShowRulers() 414
- getShowStatusBar() 423
- getShowSyntaxColoring() 414
- getShowTableBorders() 414
- getShowToolbar() 415
- getShowToolTips() 429
- getShowTracingImage() 415
- getShowWordWrap() 415
- getSiteRoot() 332
- GetSiteRootForFile() 125
- getSiteRootForFile() 118
- getSites() 359
- getSnapDistance() 451
- getSnapToGrid() 416
- getSPColumnList() 153
- getSPColumnListNamedParams() 154
- getSPParamsAsString() 155
- getStepCount() 296
- getStepInstrument() 66
- getStepsAsJavaScript() 297
- getStyles()
 - dreamweaver.cssStylePalette.getStyles() 242
 - dreamweaver.htmlStylePalette.getStyles() 302

- getSystemFontList() 286
- getTableExtent() 396
- getTables() 156
- getText() 141, 378
- getTextAlignment() 272
- getTextCallback() 142
- getTextFormat() 273
- getTokens() 389
- getTracingImageOpacity() 435
- getTranslatedAttribute() 20
- getTranslatorList() 432
- getUndoState() 298
- getUndoText() 292
- getUserName() 156
- GetVersionName() 125
- getVersionName() 119
- GetVersionNum() 126
- getVersionNum() 119
- getView() 448
- getViews() 157
- getWindowTitle() 448



- hasCharacterMarkup() 273
- hasChildNodes()
 - for comment objects 22
 - for document objects 19
 - for tag objects 20
 - for text objects 22
- hasTracingImage() 475
- hasTranslatedAttributes() 20
- helper functions, in behaviors 89
- hidden (field) object 14
- hotspot functions 316
- htmlInspector.getShowAutoIndent() 423
- htmlInspector.getShowHighlightInvalidHTML() 424
- htmlInspector.getShowLineNumbers() 424
- htmlInspector.getShowSyntaxColoring() 424
- htmlInspector.getShowWordWrap() 425
- htmlInspector.setShowAutoIndent() 425
- htmlInspector.setShowHighlightInvalidHTML() 425
- htmlInspector.setShowLineNumbers() 426
- htmlInspector.setShowSyntaxColoring() 426
- htmlInspector.setShowWordWrap() 426



- identifyBehaviorArguments() 95
- image (field) object 14
- image map functions 316
- image object 14
- importXMLIntoTemplate() 253
- increaseColspan() 397
- increaseRowspan() 397
- indent() 274
- indentTextView() 378
- InfoPrefs 125
- initialPosition() 81
- initialTabs() 82
- innerHTML property 20
- insert() 379
- insertHTML() 274
- insertLibraryItem() 324
- insertObject() 275
- insertTableColumns() 398
- insertTableRows() 398
- insertText() 276
- inspectBehavior() 97
- inspectSelection() 73
- instrumentDocument() 304
- invertSelection() 359
- isColumnAutostretch() 444
- isCommandChecked() 49
- isDesignViewUpdated() 372
- isDocumentInFrame() 266
- isDomRequired() 34, 41
- isRecording() 482
- isReporting() 336
- isSelectionValid() 373
- item() 14
- itemInfo struct 188



- JavaBeans 161
- JavaScript Controls 24
- JavaScript Debugger, How the JavaScript Debugger
 - Module Works 62
- JavaScript Debugger Module API 64
- javascript URLs 23
- JS_BooleanToValue() 198
- JS_DefineFunction() 194
- JS_DoubleToValue() 197
- JS_ExecuteScript() 201

- JS_GetArrayLength() 199
- JS_GetElement() 200
- JS_IntegerToValue() 198
- JS_NewArrayObject() 199
- JS_ObjectToValue() 198
- JS_ObjectType() 198
- JS_ReportError() 201
- JS_SetElement() 200
- JS_StringToValue() 197
- JS_ValueToBoolean() 196
- JS_ValueToDouble() 196
- JS_ValueToInteger() 195
- JS_ValueToObject() 196
- JS_ValueToString() 195
- JSBool 194
- JSContext 193
- JSNative 194
- JSObject 193
- jsval 194

L

- language information 18
- latin1ToNative() 390
- layer object 14
- listFolder() 134
- loadSitesFromPrefs() 348
- loadTracingImage() 435
- LocalURLToFilePath() 126
- localURLToFilePath() 119
- locateInSite() 354

M

- makeCellWidthsConsistent() 444
- makeEditable() 359
- makeNewDreamweaverFile() 360
- makeNewFolder() 360
- makeSizesEqual() 317
- Manipulating TreeControl Content 28
- markSelectionAsEditable() 325
- math object 14
- mayLaunchFireworks() 104
- menu commands
 - API 47
 - sample code 51
 - user experience 46
- mergeTableCells() 399

- minimizeRestoreAll() 452
- MM
 - TREECOLUMN 25
 - TREECONTROL 25
 - TREENODE 25
- mm_jsapi.h
 - including 193
 - sample file 202
- MM_returnValue 89
- mmcolorbutton 29
- MMDB.getColumnAndTypeList() 147
- MMDB.getColumnList() 148
- MMDB.getColumnsOfTable() 149
- MMDB.getConnectionList() 149
- MMDB.getConnectionName() 150
- MMDB.getConnectionString() 151
- MMDB.getDriverName() 151
- MMDB.getPassword() 152
- MMDB.getRuntimeConnectionType() 152
- MMDB.getSPColumnList() 153
- MMDB.getSPColumnListNamedParams() 154
- MMDB.getSPParamsAsString() 155
- MMDB.getTables() 156
- MMDB.getUserName() 156
- MMDB.getViews() 157
- MMDB.showConnectionMgrDialog() 158
- MMDB.showResultset() 158
- MMDB.showSPResultset() 159
- MMDB.showSPResultsetNamedParams() 160
- MMHttp.clearTemp() 138
- MMHttp.getFile() 138
- MMHttp.getFileCallback() 140
- MMHttp.getText() 141
- MMHttp.getTextCallback() 142
- MMHttp.postText() 143
- MMHttp.postTextCallback() 144
- MMInfo.h 121
- MMJB*() functions 161
- MMJB.getClasses() 163
- MMJB.getClassesFromPackage() 164
- MMJB.getErrorMessage() 164
- MMJB.getEvents() 162
- MMJB.getIndexedProperties() 163
- MMJB.getMethods() 162
- MMJB.getProperties() 161
- MMNotes object 116
- MMNotes.close() 116

- MMNotes.filePathToLocalURL() 117
- MMNotes.get() 117
- MMNotes.getKeyCount() 117
- MMNotes.getKeys() 118
- MMNotes.getSiteRootForFile() 118
- MMNotes.getVersionName() 119
- MMNotes.getVersionNum() 119
- MMNotes.localURLToFilePath() 119
- MMNotes.open() 120
- MMNotes.remove() 120
- MMNotes.set() 120
- moveBehaviorDown() 224
- moveBehaviorUp() 225
- moveSelectionBy() 318

N

- nativeToLatin1() 390
- navigator object 14
- newBlankTemplate() 327
- newBlock() 277
- newEditableRegion() 325
- newFromDocument() 460
- newFromTemplate() 254
- newHomePage() 361
- newSite() 361
- newStyle()
 - dreamweaver.cssStylePalette.newStyle() 243
 - dreamweaver.htmlStylePalette.newStyle() 303
- nextParagraph() 312
- nextWord() 312, 379
- node constants 18
- Node.COMMENT_NODE 18
- Node.DOCUMENT_NODE 18
- Node.ELEMENT_NODE 18
- odelist object 14
- nodes 18
- Node.TEXT_NODE 18
- nodeToOffsets()
 - dom.nodeToOffsets() 343
 - dreamweaver.nodeToOffsets() 462
- nodeType property
 - of comment objects 22
 - of document objects 19
 - of tag objects 20
 - of text objects 22
- _notes folder 116
- notifyFlashObjectChanged() 277
- notifyMenuUpdated() 329
- number object 14

O

- object object 14
- objects
 - adding to Insert menu 36
 - adding to Objects panel 36
 - API 33
 - user experience 32
- objectTag() 34
- offsetsToNode()
 - dom.offsetsToNode() 344
 - dreamweaver.offsetsToNode() 463
- onBlur 14
- onChange 14
- onClick 14
- onFocus 14
- onLoad 14
- onMouseDown 14
- onMouseOut 14
- onMouseOver 14
- onMouseUp 14
- onResize 14
- open()
 - MMNotes.open() 120
 - site.open() 362
- openDocument() 254
- openDocumentFromSite() 255
- openInFrame() 255
- OpenNotesFile() 126
- OpenNotesFilewithOpenFlags() 127
- openWithApp() 247
- openWithBrowseDialog() 248
- openWithExternalTextEditor() 248
- openWithImageEditor() 249
- optimizeInFireworks() 105
- option object 14
- outdent() 278
- outdentTextView() 380
- outerHTML property 20



- pageDown() 313, 380
- pageUp() 313, 381
- parentNode property
 - of comment objects 22
 - of document objects 19
 - of tag objects 20
 - of text objects 22
- parentWindow property 19
- password (field) object 14
- passwords 152
- playAllPlugins() 435
- playPlugin() 436
- playRecordedCommand() 293
- popupAction() 221
- popupCommand() 463
- postText() 143
- postTextCallback() 144
- previousParagraph() 314
- previousWord() 314, 381
- processFile() 57
- Property inspectors
 - API 72
 - required functions 72
 - sample code 74
 - user experience 71
- put() 362



- quitApplication() 287



- radio object 14
- read() 135
- readFile(), for Flash objects 113
- reapplyBehaviors() 218
- receiveArguments()
 - in menu commands 49
 - in regular commands 41
- recordPathOfLayer() 405
- recreateCache() 363
- recreateFromDocument() 461
- redo()
 - dom.redo() 291
 - dreamweaver.redo() 293

- referencePalette.getFontSize(),
 - dreamweaver.referencePalette.getFontSize() 215
- referencePalette.setFontSize(),
 - dreamweaver.referencePalette.setFontSize() 216
- refresh() 363
- regexp object 14
- relativeToAbsoluteURL() 332
- releaseDocument() 256
- reloadMenus() 329
- remotelsValid() 364
- remove() 120, 136
- removeAllBreakpoints() 306
- removeAllSpacers() 445
- removeAllTableHeights() 399
- removeAllTableWidths() 399
- removeAttribute() 20
- removeBehavior()
 - dom.removeBehavior() 218
 - dreamweaver.timelineInspector.remove
 - Behavior() 406
- removeCharacterMarkup() 278
- removeCSSStyle() 237
- removeEditableRegion() 326
- removeFontMarkup() 279
- removeFrame() 406
- removeKeyframe() 406
- removeLink()
 - dom.removeLink() 279
 - site.removeLink() 364
- RemoveNote() 127
- removeObject() 407
- removeSpacerFromColumn() 445
- removeTimeline() 407
- renameSelectedItem() 461
- renameSelectedTemplate() 465
- renameSelection() 365
- renameTimeline() 407
- replace() 260
- replaceAll() 261
- replaceRange() 382
- replaySteps() 298
- reportError() 67
- reportWarning() 68
- reset object 14
- resizeSelection() 280
- resizeSelectionBy() 318
- resizeTo() 14

- results.setResultData() 337
- resWin.addItem() 338
- resWin.setColumnWidths() 339
- resWin.setFileList() 340
- resWin.setTitle() 340
- resWin.startProcessing() 340
- resWin.stopProcessing() 341
- revertDocument() 256
- runCommand() 233
- runTranslator() 431



- saveAll() 257
- saveAllFrames() 267
- saveAsCommand() 299
- saveAsImage() 365
- saveDocument() 257
- saveDocumentAs() 258
- saveDocumentAsTemplate() 258
- saveFrameset() 259
- saveFramesetAs() 259
- saveSitesToPrefs() 348
- scrollLineDown() 382, 383, 384
- scrollLineUp() 383
- scrollPageDown() 383
- scrollPageUp() 384
- SCS 185
- SCS_AfterGet() 189
- SCS_AfterPut() 190
- SCS_BeforeGet() 188
- SCS_BeforePut() 189
- SCS_canCheckin() 186
- SCS_canCheckout() 185
- SCS_canConnect() 184
- SCS_canDelete() 187
- SCS_canGet() 184
- SCS_canNewFolder() 187
- SCS_canPut() 185
- SCS_canRename() 187
- SCS_CanUndoCheckout() 186
- SCS_Checkin() 176
- SCS_Checkout() 176
- SCS_Connect() 167
- SCS_Delete() 171
- SCS_Disconnect() 168
- SCS_Get() 170
- SCS_GetAgentInfo() 167

- SCS_GetCheckoutName() 175
- SCS_GetConnectionInfo() 173
- SCS_GetDesignNotes() 181
- SCS_GetErrorMessage() 179
- SCS_GetErrorMessageLength() 179
- SCS_GetFileCheckoutList() 178
- SCS_GetFolderList() 170
- SCS_GetFolderListLength() 169
- SCS_GetMaxNoteLength() 180
- SCS_GetNewFeatures() 175
- SCS_GetNoteCount() 180
- SCS_GetNumCheckedOut() 178
- SCS_GetNumNewFeatures() 174
- SCS_GetRootFolder() 169
- SCS_GetRootFolderLength() 168
- SCS_IsConnected() 168
- SCS_IsRemoteNewer() 183
- SCS_ItemExists() 172
- SCS_NewFolder() 171
- SCS_Put() 171
- SCS_Rename() 172
- SCS_SetDesignNotes() 182
- SCS_SiteDeleted() 173
- SCS_SiteRenamed() 174
- SCS_UndoCheckout() 177
- select color control for Javascript extensions 29
- select object 14
- select() 14
- selectAll()
 - dom.selectAll() 345
 - dreamweaver.selectAll() 347
 - site.selectAll() 366
- SelectAllCheckedOutFiles 495
- selectChild() 333
- selectHomePage() 366
- selectionChanged() 82
- selectNewer() 367
- selectParent() 334
- selectParentTag() 384
- selectTable() 345
- set() 120
- setActiveWindow() 452
- setAsHomePage() 367
- setAttribute() 20
- setAttributeWithErrorChecking() 280
- setAutoplay() 408
- setColumnAutostretch() 446

- setConnectionState() 368
- setCurrentFrame() 408
- setCurrentLine() 385
- setCurrentSite() 368
- setEditNoFramesContent() 416
- setFloaterVisibility() 453
- setFocus() 369
- setHideAllFloaters() 427
- setHideAllVisualAidst() 417
- setInterval() 14
- setLayerTag() 319
- setLayout() 369
- setLinkHref() 281
- setLinkTarget() 281
- setLinkVisibility() 370
- setListBoxKind() 282
- setListTag() 282
- setLoop() 409
- setMenuText() 50
- SetNote() 127
- setPreventLayerOverlaps() 417
- setPrimaryView() 453
- setRulerOrigin() 436
- setRulerUnits() 437
- setSelectedBehavior() 226
- setSelectedNode() 346
- setSelection()
 - dom.setSelection() 347
 - dreamweaver.setSelection() 464
 - site.setSelection() 370
- setShowDependents() 430
- setShowFrameBorders() 418
- setShowGrid() 418
- setShowHeaderView() 418
- setShowHiddenFiles() 430
- setShowHighlightInvalidHTML() 419
- setShowImageMaps() 419
- setShowInvisibleElements() 419
- setShowLayerBorders() 420
- setShowLayoutTableTabs() 446
- setShowLayoutView() 447
- setShowLineNumbers() 420
- setShowPageTitles() 430
- setShowRulers() 420
- setShowStatusBar() 427
- setShowTableBorders() 421
- setShowToolbar() 421
- setShowToolTips() 431
- setShowTracingImage() 422
- setShowWordWrap() 422
- setSnapDistance() 454
- setSnapToGrid() 422
- setTableCellTag() 400
- setTableColumns() 400
- setTableRows() 401
- setTextAlignment() 283
- setTextFieldKind() 283
- setTimeout() 14
 - in floating panels 83
 - use with FWLaunch 106
- setTracingImageOpacity() 438
- setTracingImagePosition() 437
- setUndoState() 300
- setUpComplexFind() 261
- setUpComplexFindReplace() 262
- setUpFind() 263
- setUpFindReplace() 264
- setView() 448
- showAboutBox() 288
- showConnectionMgrDialog() 158
- showFindDialog() 265
- showFindReplaceDialog() 265
- showFontColorDialog() 284
- showGridSettingsDialog() 440
- showInsertTableRowsOrColumnsDialog() 401
- showListPropertiesDialog() 282
- showPagePropertiesDialog() 290
- showPreferencesDialog() 288
- showProperties() 454
- showQuickTagEditor() 335
- showReportsDialog() 336
- showResultSet() 158
- showSPResultSet() 159
- showSPResultSetNamedParams() 160
- shutdown commands 23
- Shutdown folder 23
- site object
 - methods of 206
 - properties of 18
- site.addLinkToExistingFile() 349
- site.addLinkToNewFile() 349
- site.browseDocument() 485
- site.canAddLinkToFile() 486
- site.canChangeLink() 486

- site.canCheckIn() 487
- site.canCheckOut() 487
- site.canConnect() 488
- site.canFindLinkSource() 488
- site.canGet() 488
- site.canLocateInSite() 489
- site.canMakeEditable() 489
- site.canMakeNewFileOrFolder() 490
- site.canOpen() 490
- site.canPut() 491
- site.canRecreateCache() 491
- site.canRefresh() 492
- site.canRemoveLink() 492
- site.canSelectAllCheckedOutFiles() 493
- site.canSelectNewer() 493
- site.canSetLayout() 492
- site.canSynchronize() 494
- site.canUndoCheckOut() 494
- site.canViewAsRoot() 494
- site.changeLink() 350
- site.changeLinkSitewide() 350
- site.checkIn() 351
- site.checkLinks() 351
- site.checkOut() 352
- site.checkTargetBrowsers() 352
- site.defineSites() 353
- site.deleteSelection() 353
- site.findLinkSource() 354
- site.get() 355
- site.getCheckOutUser() 355
- site.getCheckOutUserForFile() 356
- site.getConnectionState() 356
- site.getCurrentSite() 357
- site.setFocus() 357
- site.getLinkVisibility() 358
- site.getSelection() 358
- site.getShowDependents() 428
- site.getShowHiddenFiles() 428
- site.getShowPageTitles() 429
- site.getShowToolTips() 429
- site.getSites() 359
- site.invertSelection() 359
- site.locateInSite() 354
- site.makeEditable() 359
- site.makeNewDreamweaverFile() 360
- site.makeNewFolder() 360
- site.newHomePage() 361
- site.newSite() 361
- site.open() 362
- site.put() 362
- site.recreateCache() 363
- site.refresh() 363
- site.remoteIsValid() 364
- site.removeLink() 364
- site.renameSelection() 365
- site.saveAsImage() 365
- site.selectAll() 366
- site.SelectAllCheckedOutFiles() 495
- site.selectHomePage() 366
- site.selectNewer() 367
- site.setAsHomePage() 367
- site.setConnectionState() 368
- site.setCurrentSite() 368
- site.setFocus() 369
- site.setLayout() 369
- site.setLinkVisibility() 370
- site.setSelection() 370
- site.setShowDependents() 430
- site.setShowHiddenFiles() 430
- site.setShowPageTitles() 430
- site.setShowToolTips() 431
- site.synchronize() 371
- site.undoCheckOut() 371
- site.viewAsRoot() 372
- snapTracingImageToSelection() 438
- splitFrame() 267
- splitTableCell() 402
- SQL statements
 - getting columns from 147, 148
 - showing results of 158
- startBlock() 68
- startDebugger() 307
- startOfDocument() 315, 385
- startOfLine() 315, 386
- startRecording() 294
- startup commands 23
- Startup folder 23
- status codes 137
- statusCode property 137
- stopAllPlugins() 439
- stopPlugin() 439
- stopRecording() 294

- stored procedures 146
 - getting columns from 153, 154
 - getting parameters for 155
 - showing results of 159, 160
- string object 14
- stripTag() 334
- submit object 14
- SWFFile.createFile() 110
- SWFFile.getNaturalSize() 111
- SWFFile.getObjectType() 112
- SWFFile.readFile() 113
- synchronize() 371
- synchronizeDocument() 387

T

- tables 156
 - getting columns of 149
- tag object 20
- tagName property 20
- text (field) object 14
- text node 22
- text object 22
- textarea object 14
- toggleFloater() 455
- topPage() 386
- translated attributes
 - finding in tags 20
- Tree Control Content, Manipulating 28
- Tree controls 24
- typeof operator 129

U

- undo()
 - dom.undo() 292
 - dreamweaver.undo() 294
- undoCheckOut() 371
- unescape() 14
- updateCurrentPage() 326
- updatePages() 327
- updateReference() 455
- URL property 19
- user names 156
- useTranslatedSource() 433

V

- validateFireworks() 106
- versioning 18
- view tables 157
- viewAsRoot() 372

W

- W3C 14
- window object 14
- windowDimensions() 59
 - in behavior actions 98
 - in menu commands 50
 - in object files 35
 - in regular commands 42
- wrapSelection() 387
- wrapTag() 335
- write() 136